Automated Test Framework For The Wireless Protocol Stack Development

by

Qing He

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2007

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

Qing He

I understand that my thesis may be made electronically available to the public.

Qing He

Abstract

Testing plays an important role in the wireless protocol stack development. In order to free the testers out of the shielded chamber, allow both the developers and the testers to use the test systems remotely and maximize the expensive test system usage. An automated test framework is highly demanded.

In this thesis, the design of the automated test framework is introduced. There are four main components in the test framework. They are the front end, scheduler, test engine and data storage. The architecture and the protocol among these components are described. Further, the evaluation of the scheduler is conducted based on the queueing theory. Based on the simulation result, a good scheduling algorithm is proposed. Compared with the original scheduling algorithm, the new algorithm improves the performance of the low priority users significantly when the test systems are limited. Moreover, the detail design of the test engine is presented. With the control of the intelligent test engine, the automated test framework has the capability to launch the test cases automatically, catch the commands sent by the test system and manipulate the SUT (System Under Test) without human's interrupt. It fulfills the objective of automation.

The automated test framework has been deployed and is working well.

Acknowledgements

I would like to take this opportunity to express my thanks to those who helped me with various aspects of developing the automated test framework and the writing of this thesis.

First and foremost, I would like to express my deep gratitude to my supervisor Professor Xuemin (Sherman) Shen for his continual guidance, encouragement and support during my graduate studies. I have benefited tremendously from his supervision and sharing with me his wealth of knowledge in the area of wireless communications.

Moreover, I would also like to thank Professor Pin-Han Ho and Professor Liang-Liang Xie for reviewing this thesis and for their insightful comments and suggestions.

In addition, many thanks to my managers Michael Doub and Lara Swift who gave me the instruction and the opportunity to work on this project. I am grateful to all my co-workers, especially Peter Xie and Will Huang for their friendship, developing discussion and support. Thanks to the administrative support staff, Wendy Boles and Karen Schooley, for their attention and assistance.

Finally, special thanks are due to my husband and parents for their love and support, without which I would never fulfill my dream and finish my Master program.

Dedication

This thesis is dedicated to my husband and parents.

Table of Contents

Chapter 1 Introduction 1
1.1 The testing in the wireless protocol stack development 1
1.2 The testing architecture 2
1.3 Motivation 5
1.4 Project objective and expected contributions 6
1.5 Main challenges 7
1.6 Thesis outline 8
Chapter 2 Automated Test Framework Architecture 9
2.1 Overview 9
2.1.1 Front End 9
2.1.2 Scheduler 11
2.1.3 Test Engine 12
2.1.4 Data Storage 14
2.2 The protocol among the Front End, Scheduler and Test Engine 14
2.2.1 The language of the protocol 14
2.2.2 The protocol design 15
Chapter 3 Scheduler Evaluation 21
3.1 Characteristics of queueing processes 21
3.2 Basic Model 23
3.3 Simulation 25
3.3.1 Input distribution 26
3.3.2 Bookkeeping 27
3.3.3 Output analysis 27
3.4 Results 29
Chapter 4 Test Engine Design 36
4.1 Test Manager 36
4.1.1 The console design of the Test Manager 36
4.1.2 The state machine design of the Test Manager 38
4.2 Automator 40
4.2.1 The console design of the Automator 40

4.2.2 The Automator architecture 42
4.2.3 Message Handler design 43
Chapter 5 Conclusion 47
Bibliography 48

List of Figures

Figure 1-1: The testing architecture 3
Figure 1-2: The testing architecture of interoperability testing 4
Figure 1-3: Separate the interoperability testing architecture into two conformance testing architectures 4
Figure 1-4: Automated Test Framework component structure 7
Figure 2-1: Automation Test Framework architecture 10
Figure 2-2: Test Engine structure 13
Figure 3-1: Basic Model 25
Figure 3-2: Simulation flow chart 28
Figure 3-3-1: The mean waiting time in queue vs. the number of test systems 33
Figure 3-3-2: The mean number of TCs in queue vs. the number of test systems 34
Figure 3-3-3: The service utility vs. the number of test systems 35
Figure 4-1: Test Manager Console 37
Figure 4-2: The state machine of the Test Manager 39
Figure 4-3: Automator Console 41
Figure 4-4: Automator architecture 42
Figure 4-5: Message Handler flow chart 46

List of Tables

Table 3-1: Simulation input data table 30
Table 3-2: Simulation output data table – scheduling algorithm I 31
Table 3-3: Simulation output data table – scheduling algorithm II 32

Chapter 1
Introduction

Testing plays an important role in the wireless protocol stack development. Establishing an efficient test environment for the regular regress testing is key to achieve successful, reliable, and predictable wireless protocol stack. Regression testing provides the only reliable means to verify that code base changes and additions do not break an application's existing functionality, and it can have the single greatest impact in controlling product release delays, budget overruns, and the prospect of software errors slipping into released/deployed products. Early identification of problems introduced by code modification can save countless hours of development time and allows the development team to maintain and modify the wireless protocol stack without fear of breaking previously-correct functionality. Yet development organizations often give up on the full regression testing because they find it complicated and time consuming. So an automated test framework is highly demanded.

1.1 The testing in the wireless protocol stack development

Three types of testing have to be done during the wireless protocol stack development. They are internal testing, conformance testing and interoperability testing.

Usually, there are two teams in the wireless protocol stack development group. One is the developing team whose responsibility is coding the protocol stack based on the ETSI specifications. The other team is the testing team whose responsibility is developing the test scripts based on the ETSI specifications and the feature list provided by the developing team to discover the defects in the protocol stack software and report to the developing team to find a good solution to correct the issues. These two teams are working very closely during the protocol stack development, especially before the protocol stack gets mature. The testing done in this phase by the testing team is called internal testing. The internal testing includes the unit testing, integration testing, and system testing. The unit testing is the process of testing an individual software unit, be it a class, function or module, to evaluate whether it performs the required functions and returns the correct results and data. Unit testing is white box testing, in that the knowledge of the internal working of the code is required. Usually, the developers are involved. Integration testing follows the unit testing and precedes system testing. Integration testing takes as its input modules that have been unit tested, group them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output. The integrated system is ready for the system testing. The system testing is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirement. It

falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic. It seeks to detect defects both within the “inter-assemblages” and also within the system as a whole. A network simulator is configured and developed for different test cases. These test cases are designed according to the system functional requirement specification. Regression test is required during the development.

After the matured protocol stack is released from the wireless protocol stack development group, the official testing is started. The conformance and interoperability testing are both important and useful approaches to the testing of standardized protocol implementations although it is unlikely that one will ever fully replace the other. Conformance testing is able to show that a particular implementation complies with all of the protocol requirements specified in the associated base standard. ETSI, ITU, 3GPP and other standardization bodies develop conformance test suites. Vendors of telecom equipments or operators (the customers) are used to apply these conformance test suites to show conformance of the products or for type approval. However, it is difficult for such testing to be able to prove that the implementation will interoperate with similar implementations in other products. It may happen that the interoperability test of two implementations fails even if they passed the conformance test. On the other hand, interoperability testing checks if two different implementations of the same protocol have the capability of inter-working. It can clearly demonstrate that two implementations will cooperate to provide the specified end-to-end functions, but can not easily prove that either of them conforms to the detailed requirements of the protocol specification. As most protocol and interface specifications are written in natural languages, such as English and French, and they are subject to different interpretations. There can be common errors in implementations or misunderstandings of the protocol description. If the two implementations have the same error of this kind, this error will not be discovered by the interoperability testing. Instead, the conformance testing is able to catch this kind of errors. Usually, the interoperability test is done after the conformance tests. Passing the conformance testing and passing the interoperability testing are the preconditions to put a wireless product on the market. Regression conformance and interoperability testing is needed when the new features are implemented or bugs are fixed.

1.2 The testing architecture

Figure 1-1 shows the testing architecture for the system testing and conformance testing. It consists three parts, the tester, the SUT (System Under Test) and the test system.

The tester is a person who is responsible for configuring the test system, launching the test cases, collecting the test results, as well as manipulating the SUT.

The SUT stands for the System Under Test. It is a wireless product developed by the different manufactures based on the standard specifications.

The test system is a network simulator which is a radio-based test equipment. It provides a comprehensive solution for the base station emulation. A variety of cell site configuration options are available for each of the wireless formats. This enables the user to establish a network connection, such that the SUT believes it is on a real network. It connects with SUT by an RF cable. Moreover, the test system provides the programmable interface for the testers to develop test cases. The conformance test cases are specified by the ETSI MS conformance specification and developed by the test system venders. These test cases must be certified before put into use. The conformance testing covers most features defined in the ETSI specifications. But for the uncovered parts, the SUT manufactures use the opened programmable interface of the test system to develop their own test cases for the internal testing.

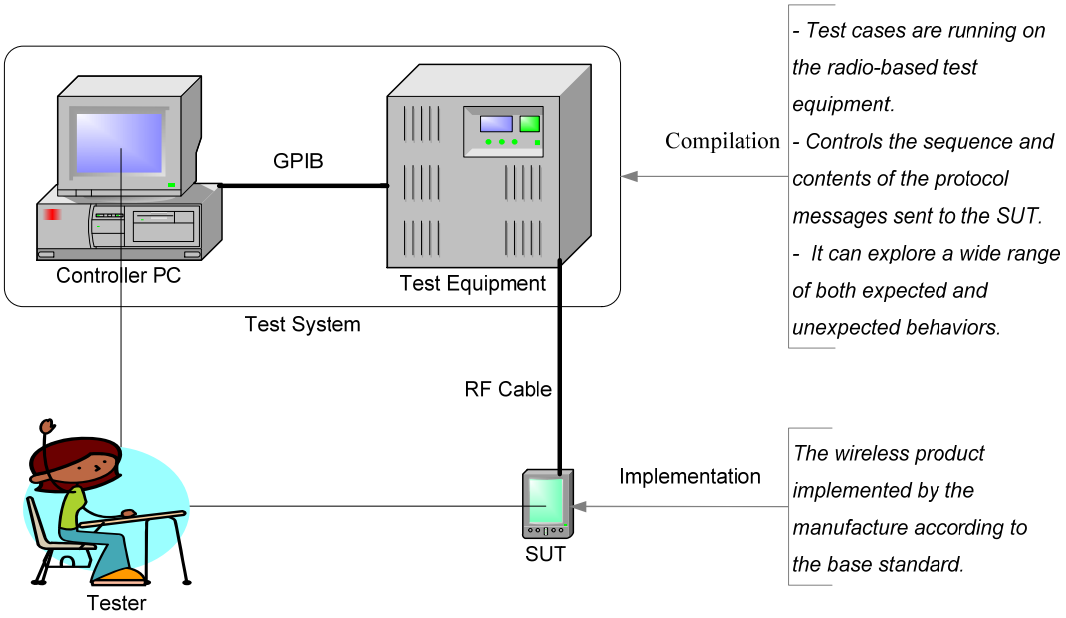


Figure 1-1: The testing architecture

Figure 1-2 shows the architecture of the interoperability testing. The essential differences of interoperability testing from conformance testing are that the target of testing is two SUTs and the behavior to be expected for testing should be inferred from the respective specifications of two SUTs. It uses the configurable real network equipments instead of the network simulator. The interoperability test cases are the detail set of instructions that need to be taken in order to perform the test. So unlike the conformance testing where the test driver is the test scripts running on the test system, in interoperability testing, the test driver is a human operator. It tests the end-to-end functionality between two communicating systems.

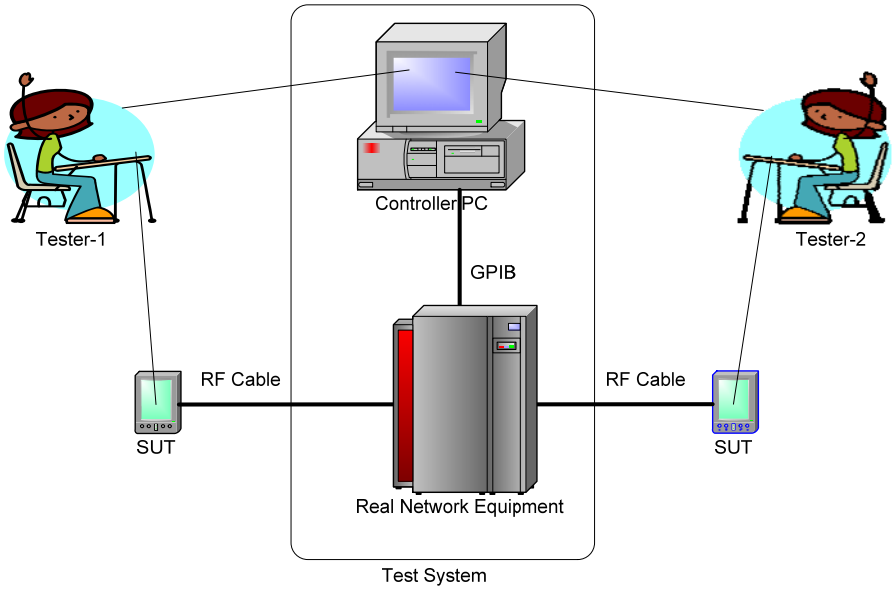


Figure 1-2: The testing architecture of interoperability testing

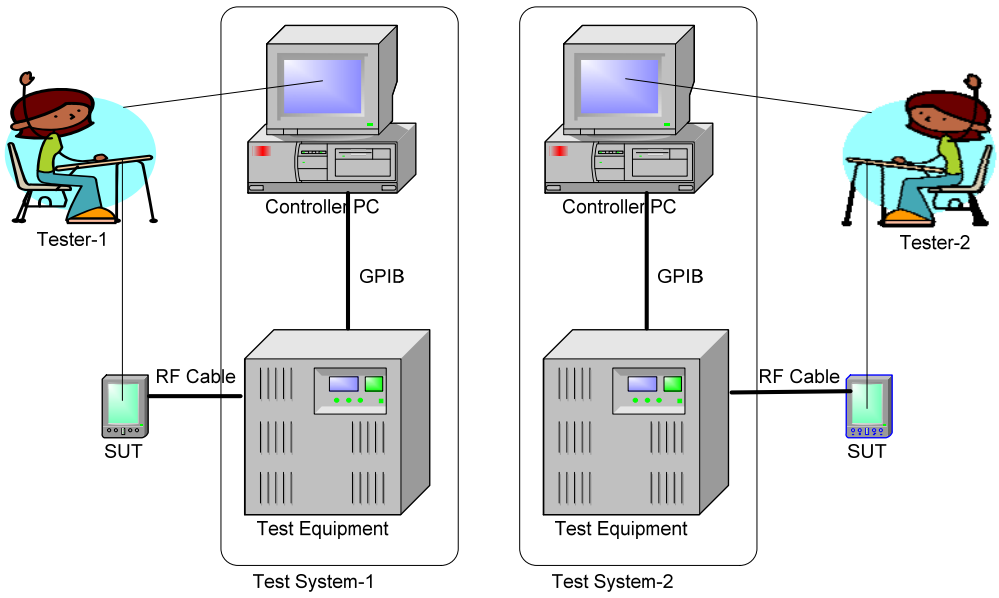


Figure 1-3: Separate the interoperability testing architecture into two conformance testing architectures

The configurable real network equipments are only available in the carrier's lab. If a test case fails, sometimes the developers have to travel to the carrier's lab for the trouble shooting. It is expensive and time consuming. So the manufactures are seeking a good way to reproduce the test scenario on the test system. As the test system opens the programmable interface to control itself, the developers of the manufactures can separate the interoperability testing architecture into two conformance testing architectures as figure 1-3 shows. Based on the interoperability testing scenario, the developers implement the test cases on the test system to control the test sequences. In this way, one interoperability test case can be divided into a pair of two different test cases like conformance testing.

According to the previous analysis, the testing architecture in figure 1-1 is widely used in the wireless protocol stack development. There are two kinds of test cases. One is used by the conformance testing and the test cases are developed by the ETSI and certified. The other one is for the system testing and reproducing the test scenarios of interoperability testing. This kind of the test cases is implemented by the manufactures for the internal testing.

1.3 Motivation

Testing is an integral part in the wireless protocol stack development. It is broadly deployed in every phase during the development. Typically, more than 50% of the development time is spent in testing. The testing architecture described in figure 1-1 is used in the system testing, conformance testing and also involved in the interoperability testing.

First, the amount of the test cases developed in the test system of figure 1-1 is large. There are thousands of conformance test cases developed by the ETSI. It covers almost all the wireless technologies like GSM, GPRS, EDGE, UMTS and so on. The multi-band SUT has to pass the same test case on all the supported bands. For example, if the SUT supports GSM900, DCS1800, PCS1900 and GSM850 four bands, the same test case has to be run four times. ETSI is working on developing more test cases as more new features are added into the specification. Besides the large amount of the conformance test cases, the developers also implement many test cases with the scenarios that out of the coverage of the conformance testing for the internal testing. Some test cases, especially the performance measurement test cases, are very long which may last several hours.

Second, the demand of the test system described in figure 1-1 is large. Usually several products are developed paralleled in a big company. As passing the conformance testing and interoperability testing are the precondition to put the wireless products on the market, all of the products under developing need the testing time. The error debugging and regression testing occupy huge testing time. For the failed test cases, the tester has to take the logs, send to the developer. Once the developer fixes the bugs, the tester must verify the change. This cycle is continuous till the issue is

resolved. Once the source code is changed, either because of the bug fix or the new feature implementation, the regression test has to be underway to make sure the new changes are good and not break any existing working features.

Third, the test systems are very expensive. The manufactures have to effectively use these test resources.

Fourth, the testing has to be done in the shielded chambers. In order to avoid the life network interference and the radio signal pollution, the test systems are in the shielded chambers. So the responsible testers have to stay in the chamber whole day to run the test cases again and again.

In conclusion, in order to free the testers out of the shielded chamber and to maximize the expensive test system usage, especially for the regression testing, an automated test framework is highly demanded.

1.4 Project objective and expected contributions

The principle objectives that must be achieved by the automated test framework are

- Reduce the number of person hours spent in the shielded chamber for manual testing
- Allow more efficient utilization of the test systems
- Reduce the overall time to complete the entire conformance testing through both parallel and continuous execution
- Execute a complete pass through all the conformance test cases on a product in a couple of weeks.
- Provide a friendly web-based user interface for the users (both developers and testers) to remotely load code on the SUT, submit a test campaign to the test system for execution, capture the test logs without requiring physically present in the shielded chamber and monitor the test progress at their own desks.
- Set up a reporting system that allows the incremental tracking of the test results over a series of incremental software builds.
- Allow for the incremental addition of additional test systems
- The ideal situation is that the system can keep running.

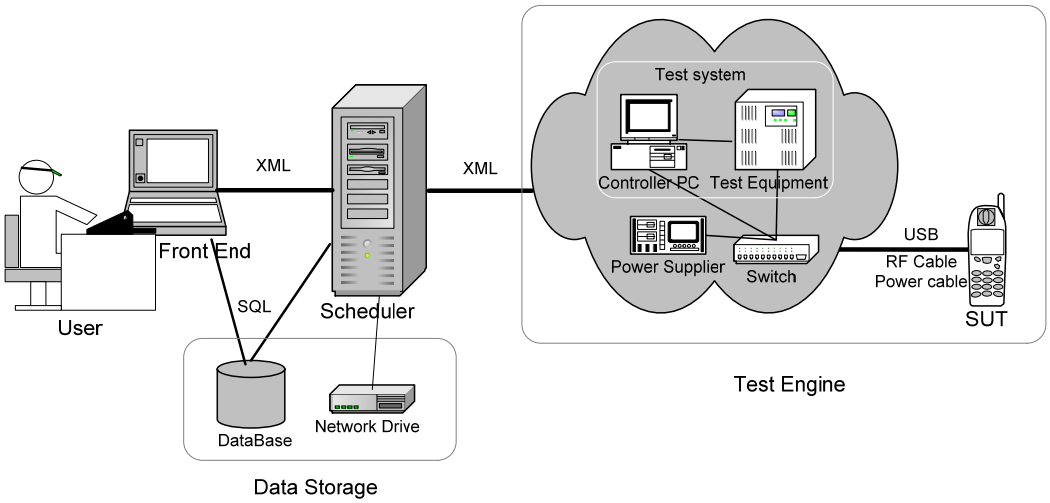


Figure 1-4: Automated Test Framework component structure

In order to fulfill the project objective, the automated test framework is designed to consist of the following components. Figure 1-4 shows the component structure.

- Front End

It is a web-based user interface running in the user's browser and allows the users to submit test campaigns, monitor system status, collect test logs and trace the test history at their desk.

- Scheduler

Assign the available test engines to the pending test campaign and return the test results.

- Test Engine

It is the automated test system showed in figure 1-1. It is responsible for executing actions received from the Scheduler and reporting corresponding results. It can configure the test equipments and launch the test cases automatically.

- Data Storage

Store the test campaign information, test results in a database and network drive.

1.5 Main challenges

The first challenge is the scheduler design. Scheduler is the central controller of the automated test framework and affects the whole system's performance. It connects the front end and the test engine. When the test campaigns submitted by the front ends enter the input queue of the scheduler, the

scheduler has to arrange the position for each test campaign based on its character. Then according to the deployed scheduling algorithm, assign the available test systems to the waiting test campaigns. A suitable scheduling algorithm needs to be designed for this automated test framework according to its specialty. To verify the performance of the different algorithms, simulation is the best way to achieve it.

The second challenge is the test engine implementation. The test engine is the core of the automated test framework. In order to free the testers, a mechanism must be designed to replace the tester's eyes and hands. In other word, the test engine should have the ability to catch the commands sent by the test system. Meanwhile, it can translate the commands into the format that the SUT understands. Then the SUT can do the required activity. In addition, the test engine should be capable of correctly configuring the test system and launching the test cases as per the user's requirement. After the test finishes, it must manage the test results and test logs correctly and communicate with the scheduler to let the user know the final test results and the location of the test logs. Moreover, the test engine should have the error recovery capability to keep both the test system and the SUT running.

Last but not the least one is the high performance requirement. The test results produced by the automated test framework must be reliable. As it is a replacement of manual testing, it must guarantee the accuracy. Otherwise, the automated test framework has no value. The other performance requirement is reporting the system status frequently, especially the critical errors. So the ability to catch errors is a challenge task. Following it, for the recoverable errors, the system should have the capability to set itself back to the right state and keep running. For the critical errors that can not be recovered, the system should report it to the administrator and set itself to the idle state. The overall goal is maximizing the usage of the test systems.

1.6 Thesis outline

The rest of the thesis is organized as follows: chapter 2 will provide the automated test framework architecture. The performance evaluation of the scheduler is studied in chapter 3 using queueing theory and a good scheduling algorithm is proposed according to the simulation result. The detail test engine design is given in chapter 4. Finally, chapter 5 presents concluding remarks.

Chapter 2
Automated Test Framework Architecture

2.1 Overview

The automated test framework consists of four major components. They are Front End, Scheduler, Test Engine and Data Storage. Figure 2-1 shows it.

2.1.1 Front End

The Front End is a web-based user interface running in the user's browser and allows users to submit and monitor test campaigns along with system status.

The Front End is responsible for the following tasks.

2.1.1.1 Campaign management

Test campaign is a group of the test cases. It consists of the following information.

- The test case numbers in the test campaign.
- The SUT type and the corresponding software build that the test campaign tests for.
- The project that the test campaign belongs to.
- The test campaign priority.
- The location to store the test logs

The test case information, the available SUT types and the corresponding software build information are obtained from the database.

The user can generate a test campaign, load the existing test campaign and delete a test campaign.

2.1.1.2 Query function

Query is one of the main functions provided by the Front End. User can get the system status, project summary, project statistics and project progress via the Front End.

System status tells user the current scheduled campaigns and the status, like active, pending. Besides the campaign status, it also shows the available test systems and the status, like running test, idle, respectively. The queue status of each test system is also trackable. Using this function, user knows the whole system status and the roughly estimated waiting time.

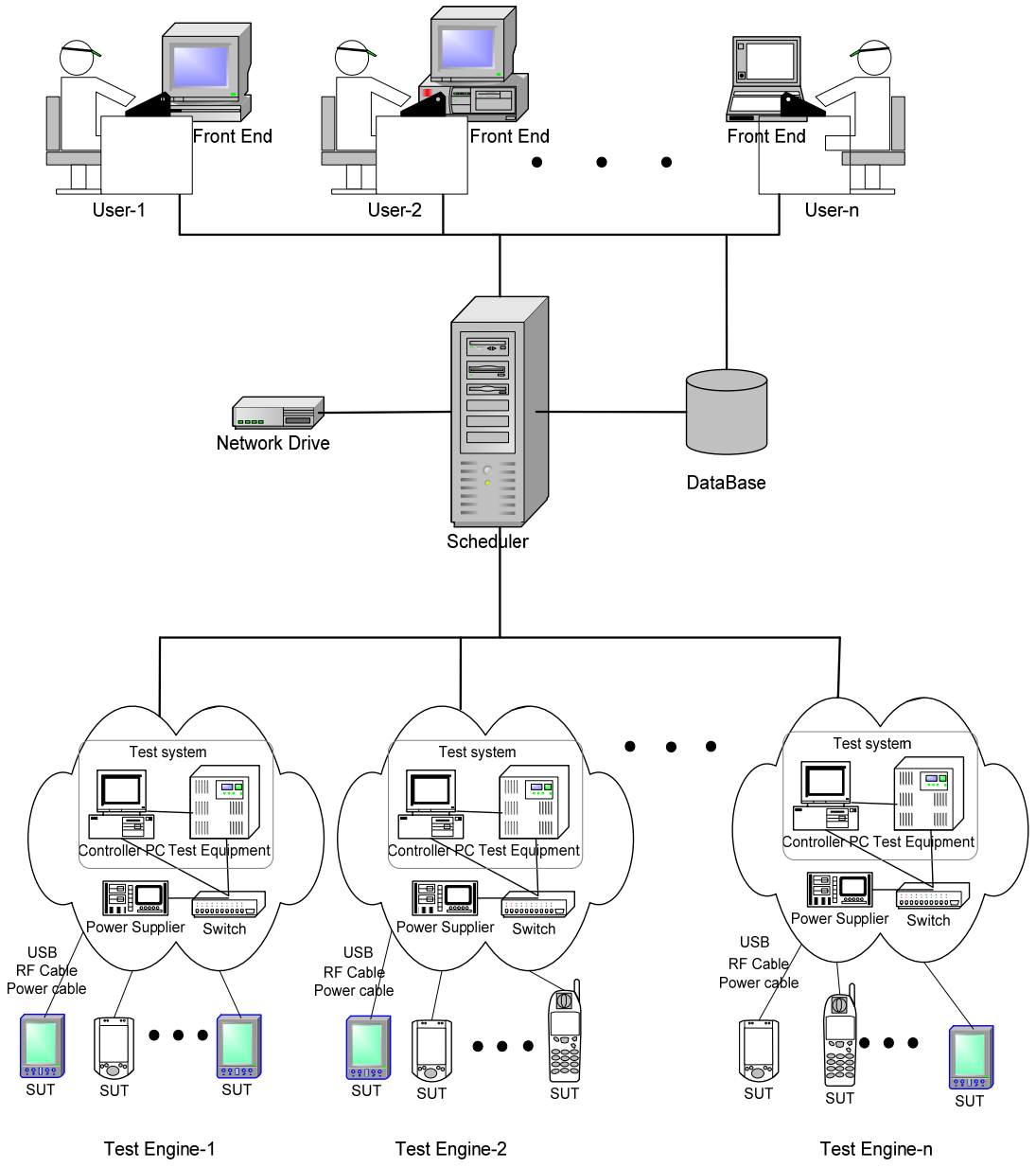


Figure 2-1: Automation Test Framework architecture

Project summary is a very useful feature for the users to track the test case results. After selecting a project, the user can check the test case list, the test case information which includes all the test results of the selected test case and the corresponding logs.

Project statistics is a great tool for the managers to track the current status of the project. It shows how many percentages of the test cases have passed and how many percentages of the test cases have failed or have not got a chance to run. It uses the graph to show the percentage of each part and is updated automatically to keep the results up to date.

Project progress is also for the project management. It tells the user how many test cases have passed or failed or not run everyday. Based on this information, the project manager can adjust the project plan and have more confidence to schedule the project release.

As some test cases like SIM test are not automatable and have to be run manually, this system can also record the information for the manual test results in the same way as the automation test results except that the manual test results are input manually.

2.1.1.3 Configure the system

This function is only opened to the system administrator. The administrator can add or delete the users into or from the system. To control the access to the automated test framework, each user has its own ID and password. Besides the access control, the administrator is responsible for maintaining the system.

2.1.2 Scheduler

The Scheduler effectively assigns the available test systems to the pending test campaigns. It sits between the Front End and the Test Engine.

The Scheduler is responsible for the following tasks.

2.1.2.1 Scheduling

Scheduling is the main task of the Scheduler. It processes the Front End requests in the priority queues. Based on the current test system status, it schedules the task on the available test systems. The scheduling algorithm design is based on the best performance of the framework.

2.1.2.2 Service to Front End

- Receive the requests from the Front End

The requests include the submit campaign request, cancel campaign request and various query requests. After receiving these requests, the Scheduler handles them according to the current status.

- Send the results back to the Front End

The results are corresponding to the requests which include the campaign result and query result. Besides these, the Scheduler informs the Front End the current status of the Test Engines.

2.1.2.3 Service to Test Engine

- Send the requests to the Test Engine

The Front End sends a test campaign to the Scheduler which could include many test cases. While under the control of the scheduling algorithm, the Scheduler sends the test cases one by one to the available test systems. Before running a test case, the Test Engine is responsible to load the required code into the SUT. So the load request is one of the requests sent to the Test Engine besides execute request, query request.

- Receive the results back from the Test Engine

Besides receiving the results corresponding to the requests, the Scheduler also receives the Test Engine status reports like the connection indication, heart beat indication, normal disconnect indication and send email indication.

2.1.3 Test Engine

The Test Engine controls the test equipments and the connected SUTs. It is responsible for executing actions received from the Scheduler and reporting corresponding results. It can configure the test equipments and launch the test cases automatically. During the test running, it can catch the commands from the test equipment, drive the SUT to do the required tasks and keep both SUT and the test equipment in the correct state.

From hardware point of view, Test Engine consists of the following components. Figure 2-2 shows the detail. The switch uses RS-232 serial cable to connect to the controller PC. One terminal of RF cable is connected to the one port of the switch and another terminal is connected to the SUT. The connection of the USB cable is the same. The power supplier is connected to the power output of the same port in the switch. It is for the fake battery.

From software point of view, Test Engine consists of two components, Test Manager and Automator. They reside in the controller PC of the test system.

2.1.3.1 Test Manager

Test Manager interacts with the Scheduler, the Automator and test equipment. It takes control of launching the test cases on the test equipment. During the test case running, it monitors the errors and when the error occurs, it takes action to handle it.

2.1.3.2 Automator

Automator interacts with the Test Manager, SUT, test equipment and switch. During the test, it translates the commands sent from the test equipment to the SUT understandable AT commands. Then sends the AT commands to the SUT to let it perform the required action. The switch controller is also implemented in the Automator.

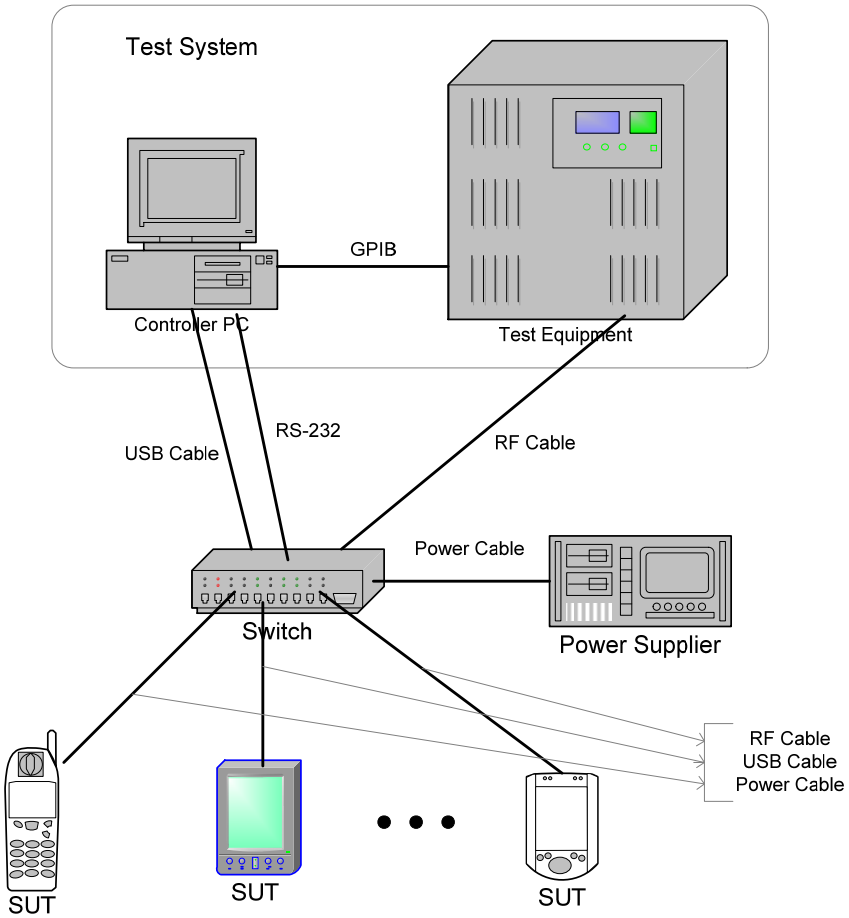


Figure 2-2: Test Engine structure

2.1.3.3 Other assistant components

- Switch for RF, Power and USB

This equipment is designed for this automated test framework only. It has 12 ports. Each port has an RF output, a Power output and a UBS connector. It connects the test equipment via RS-232 serial port. In order for the users to control the switch automatically, it opens the source code of the driver. If all the 12 ports of the switch are used, there are 12 SUTs can connect to the test equipment at the same time. As the test equipment can only test one SUT at a time, the user needs to implement a program to select and control the active port. This function is embedded in the automator.

- Fake battery with power supplier

As we know, the battery life is limited. But in this automated test framework, the battery life is assumed unlimited and keeps the SUT working all the time. So a fake battery is designed and it uses the power supplier to get energy. It is a smart proposal and works very well.

2.1.4 Data Storage

It uses database to store the test case information, test results. A network drive contains the test logs, the SUT codes to be loaded. The administrator of this component keeps the information up to date and cleans the old files on the network drive.

2.2 The protocol among the Front End, Scheduler and Test Engine

2.2.1 The language of the protocol

The Front End, Scheduler and Test Engine are implemented on different platforms by different programming languages. It is desirable to work with a common interface protocol among them.

The Extensible Markup Language (XML) is a general-purpose markup language. It is classified as an extensible language, because it allows the users to define their own tags. The primary purpose is to facilitate the sharing of data across different information systems. It is simplified subset of the Standard Generalized Markup Language (SGML), and is designed to be relatively human-legible. By adding semantic constraints, application languages can be implemented in XML.

Due to its simplicity, extensibility and wide support across available development platforms, XML is proposed to be the language for the interface protocol.

2.2.2 The protocol design

The XML interface among the framework's components is defined by three basic tags: Action, Query and Info.

An action is a command to execute a task such as "submitting a test campaign" or "loading code on a SUT". Here is an example of the XML interface with action tag.

```
<atf version = "versionNumber" refID = "refID">
      <info type = "reportingIn" heartBeatFrq = "once every x seconds">
      <equipment  equipmentID = "equipmentID" >
                  <name> equipmentName </name>
      </equipment >
      <firmwareVersion >
                  <testmanager> testmanagerVersion </testmanager>
                  <automator> automatorVersion </automator>
      </firmwareVersion >
      <deviceType>
           <name>devicename1</name>
           <name>devicename2</name>
                 …
       </deviceType>
      </info>
</atf>
```

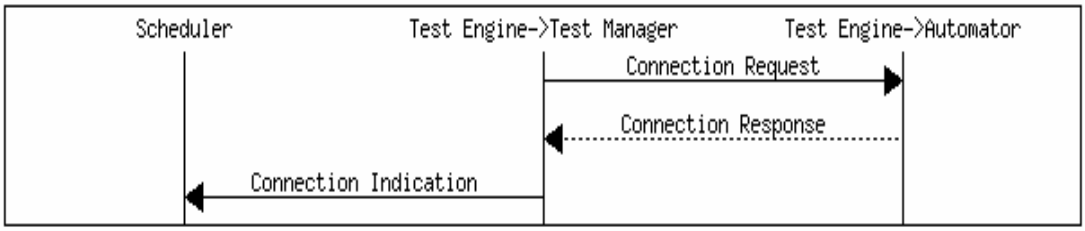
A query requests for information such as test equipment status returned in an info tag. Here is an example of the XML interface with query tag.

```
<atf version = "versionNumber" refID = "refID">
      <query type = "testcaseSupport">
      <test name="testCaseName" band="band" equipmentID="equipmentID"/>
      </query>
</atf>
```

The info tag can be also used as a notification, for example, to indicate the test equipment readiness. Here is an example of the XML interface with info tag.

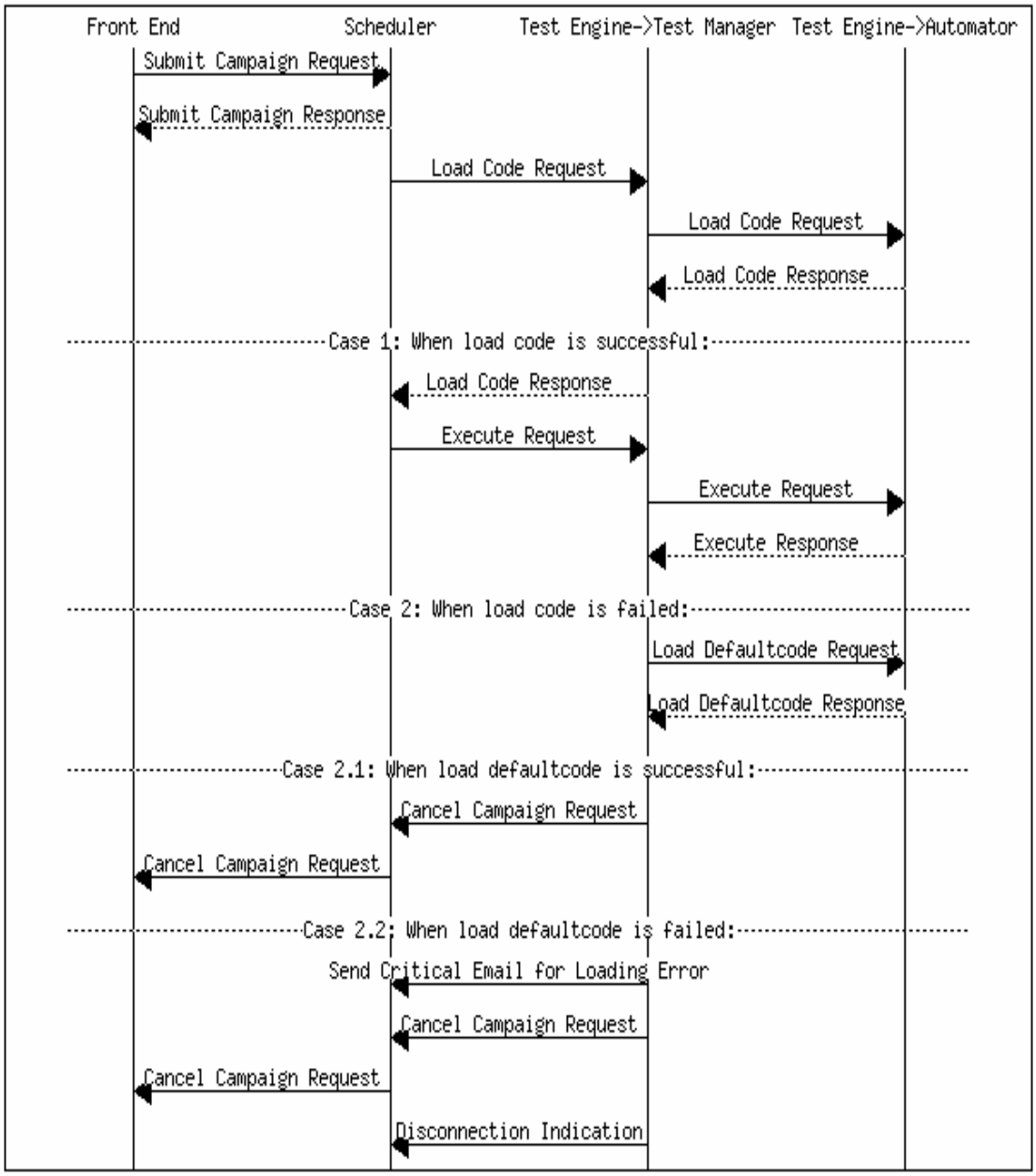
```
<atf version = "versionNumber" refID = "refID">
      <info type = "testResult">
           <test  campaignID = "n" testcaseID = "id" deviceType = "type"
                  IMEI = "imei" IMEISV = "imeisv" notes = "someString"
                  testcaseVersion="versionString">
               "Pass"|"Fail"|"Inconclusive"|"Cancelled"|" TcNotSupport"|
               " BandNotSupport"|" PlatformError"|"TcUnexecuted"
           </test>
      </info>
</atf>
```

2.2.2.1 Connection Indication



It is the first command submitted by the Test Engine when the Automated Test Framework put into use. It informs the Scheduler that the Test Engine is ready. After Scheduler gets the Connection Indication command, it stores the received information which includes the test equipment name and all the connected SUT names as well as the version of the Test Manager and Automator for the Front End to query.

2.2.2.2 Submit a campaign / Load Code / Cancel a campaign by Test Engine



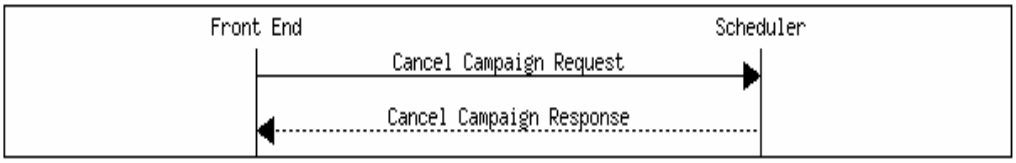
This protocol is initiated by the Front End. After receiving “Submit Campaign Request”, the Scheduler puts the campaign into the input queue and responds with “Submit Campaign Response”. On the other hand, when the Scheduler assigns an available test system to a pending campaign, it firstly sends “Load Code Request” to the Test Manager and the Test Manager routes this request to

the Automator to start the loading action. After the Automator finishes, the “Load Code Response” command is sent back to the Test Manager and Scheduler.

If the “Load Code” is successful, the Scheduler issues the “Execute Request” to the Test Manager and the Test Manager routes it to the Automator. Based on the “Execute Request” information, the Automator configures itself to the correct state and sends the “Execute Response” back. The important information in the “Execute Response” is IMEI which is the SUT identity and the Test Manager needs this value to configure the configuration file. At this time, the Test Manager configures and Launches the test case. The testing starts.

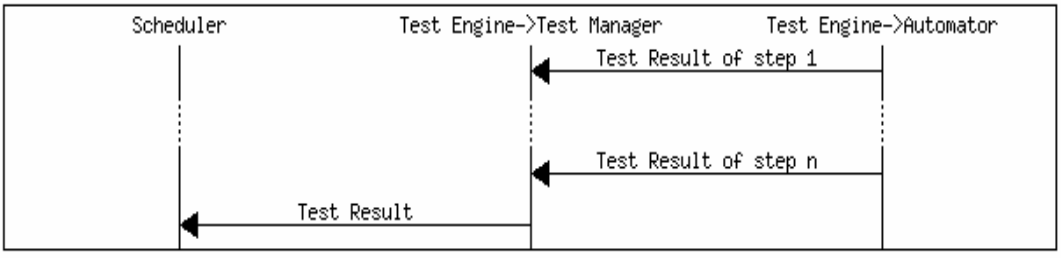
If the “Load Code” is failed, the Test Manager requests the Automator to load the default code which is a good code stored in the network drive for the error recovery. If the “Load Defaultcode” is successful, this means that the error is in the code of the campaign. So the Test Manager requests the Front End to cancel the bad campaign by “Cancel Campaign Requst” via Scheduler. In this case, the SUT is in normal state and keeps running. If the “Load Defaultcode” is also failed, this means that the SUT has unrecoverable error and the test can not keep going. So the Test Manager informs Scheduler to send a critical email to let the administrator and the user know that the system can not keep running because of the loading error. After it, the Test Manager requests the Front End to cancel the campaign by “Cancel Campaign Request” via Scheduler and sends “Disconnect Indication” to the Scheduler and enters the idle state.

2.2.2.3 Cancel a test campaign by Front End



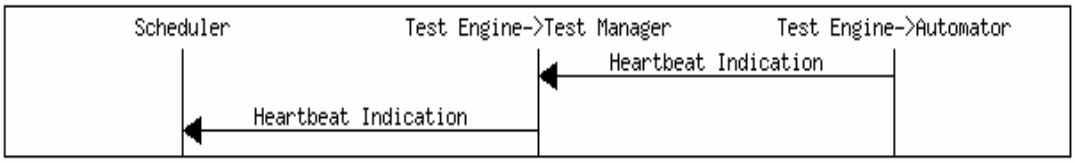
Some users can cancel the submitted campaign if the campaign is in pending state. In other word, if the campaign starts to be executed, it can not be cancelled.

2.2.2.4 Return Test Result



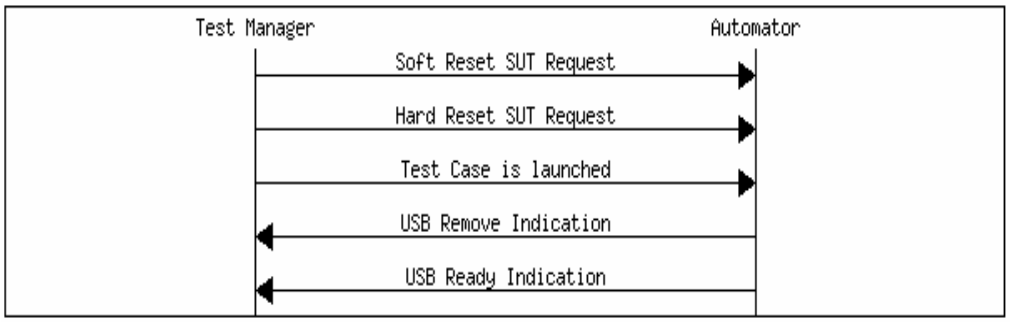
The majority of the test cases only have one step, while some test cases have several steps. After getting the test results of all the steps, the Test Manager sends the test result back to the Scheduler. Then the Scheduler stores the result to the database for the Front End to access.

2.2.2.5 Heartbeat Indication



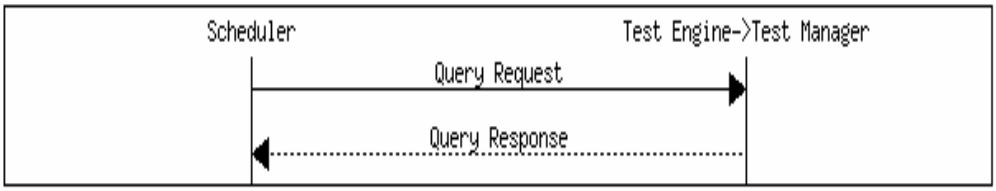
The purpose of the “Heartbeat Indication” is to inform the Scheduler that the test system is still alive. The frequency of the Heartbeat is defined in the “Connection Indication” command sent by the Automator.

2.2.2.6 Other interface commands inside Test Engine



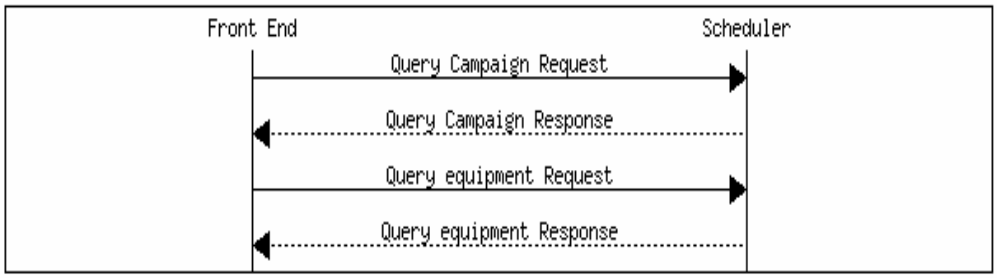
These commands are transmitted inside the Test Engine. They are used to control the SUT and inform the Test Manager the USB status.

2.2.2.7 Test case supporting query



Test case supporting query is a special feature of the Test Manager. The Test Manager keeps all the test case information of all the test equipments in the local files and guarantees the files are up to date. Therefore any PC with the Test Manager installed can talk to Scheduler for the test case supporting query. This design is to save the precious time of the test systems. As we know, the amount of the test cases is very large. To finish the query for all the test cases takes long time. With this intelligent method, this timing consuming query procedure can be done at any time when the Scheduler is idle.

2.2.2.8 Other queries



Query campaign is to get the submitted test campaigns' status, such as pending, running.

Query equipment is to get all the SUT names and status that connected to the test equipment.

Chapter 3
Scheduler Evaluation

The Scheduler plays a key role in sharing the test systems among the users in an efficient way. It assigns the available test systems to the requested users based on the test system status, test campaign characters as well as other factors. Queueing theory is used in the scheduler evaluation. A good scheduling algorithm should be based on the evaluation result.

The goal of my work is to evaluate the effectiveness of the Scheduler. It includes the average waiting time versus the number of the test systems, the idle time of the test systems, the average number of the test cases in both the high and low priority queues. From the evaluation result, I know how many test systems is suitable for the demand and the relative fairness of the Scheduler.

This chapter is organized in this way. First, the characteristics of queueing processes are introduced. Second, the basic model of the Scheduler is designed based on the characters of a queueing process. Third, the simulation implementation is presented. Finally, the simulation result is analyzed.

3.1 Characteristics of queueing processes

In most cases, there are six basic characters of queueing processes that provide an adequate description of a queueing system.

1) Arrival pattern of customers

In usual cases, the process of arrivals is stochastic and follows a particular probability distribution which describes the times between successive arrivals called *inter-arrival time*.

- Single / Batch

If the customers arrive one by one, it is called *single arrival* or if they arrive simultaneously, it is called *batch arrival*. For the batch arrival, the probability distribution of the batch size also needs to know.

- Patient / Impatient

After arrival, some customers may stay in the queue until get serviced no matter how long the queue becomes. This kind of the customers is called p*atient* customers. On the other hand, some customers leave the queue before get serviced for some reasons. They are called *impatient* customers.

- Balked / reneged / Jockey

They are for the impatient customers. Some customers leave before entering the system, they are said to have *Balked*. Whereas some customers leave after entering the system, they are said to have *reneged*. Moreover, if the system has two or more parallel waiting lines, some customers may switch from one to another, they *jockey* for positions.

- Stationary / Non-stationary

This character is used to describe the manner in which the pattern changes with time. If the arrival pattern does not change with time, it is called a *stationary* arrival pattern. Whereas, if the arrival pattern is time-depend, it is called a *non-stationary* arrival pattern.

2) Service pattern of servers

The same as the arrival pattern, the service pattern also follow a probability distribution and need to know for the simulation and evaluation.

- Single / Batch

The usual cases are one customer being served at a time by a given serve. It is called *single* service. But the cases like several customers may be served simultaneously by the same server also exist. It is called *batch* service.

- State-dependent / State-independent

If the service process depends on the number of customers waiting in the queue, it is a *state-dependent* service. Otherwise, it is a *state-independent* service.

- Stationary / Non-stationary

Like arrival pattern, if the service pattern does not change with time, it is a *stationary* service pattern. Otherwise, it is a *non-stationary* service pattern.

3) Queue discipline

The approaches to select the customers in the queue to be served, the common discipline is the first come, first serve (*FCFS*). Besides it, there are other disciplines like last come, first serve (*LCFS*), random selection for service (*RSS*), Priority (*PR*) and General discipline (*GD*).

For the PR, there are two situations, preemptive and non-preemptive. If the customer with high priority is allowed to enter service immediately even if a customer with lower priority is already in service when the high priority customer enters, this situation is called *preemptive*. Whereas, the highest priority customer can only go to the head of the queue if a customer is already in service. It can not get service until the customer in service is completed, this situation is called *non-preemptive*.

4) System capacity

If the size of the queue has limitation, it is called *finite* queueing system. Otherwise, it is called *infinite* queueing system.

5) Number of service channels

It refers to the number of parallel service stations which can serve customers simultaneously. Therefore, there are *single-channel* system and *multi-channel* system. Two types of the multi-channel system exists, they differ in that one has a single input queue, while another allows a queue for each channel.

6) Number of service stages

A queueing system may only have one stage of service which called *single-stage* of service, while some may need several steps of service to finish a task, named *multi-stage* of service.

3.2 Basic Model

Based on the character of the automated test framework and the queueing theory, we model the Scheduler by a multiserver two-queue system with two priority classes (high priority type A users and low priority type B users) of impatient users.

1) Arrival pattern of customers

- Batch

The users arrive one by one and submit one test campaign which includes several test cases to the Scheduler via Front End. Assuming the arrival process is a Poisson process. The inter-arrival time follows exponential distribution. As all the test cases of one test campaign arrive simultaneously, it belongs to the batch arrival. The probability distribution of the batch size is uniform.

- Impatient / Patient

We assume the high priority type A users are impatient and the low priority type B users are patient.

For the high priority type A users, after arrival, some of them may leave before submitting a test campaign. This kind of users is balked impatient users. Whereas some users may cancel the pending campaign they submitted after waiting a random length of time for service to begin, they are reneged impatient user. The reneging time is assumed to have exponential distribution.

For the low priority type B users, they stay in the queue until get service.

- Stationary

As the arrival pattern does not change with time, it is a stationary arrival pattern.

2) Service pattern of servers

- Single

As one test system can only run one test case at a time, it is a single service. The service time follows exponential distribution.

- State-independent

As the service process does not depend on the number of test cases waiting in the queue, it is a state-independent service.

- Stationary

As the service pattern does not change with time, it is a stationary service pattern.

3) Queue discipline

There are two classes of users. The high priority type A users and the low priority type B users. The model consists of two infinite priority queues type A and B. The test cases submitted by type A or B users are put into type A or B queue. The test cases in queue A have priority over those in the queue B in the sense that the test cases in queue A get the service first. The priority rule is non-preemptive, which simply means that once a test system is running a type B test case, it can not stop and switch to serve type A test case. Within each queue, the test cases are serviced in order of their arrival, that is, under the First Come, First Served (FCFS) discipline. In addition, we do not allow jockeying between different queues.

4) System capacity

The memory on the Scheduler PC can be very large, so we assume the size of the queue is large enough for all the pending test campaigns. It is an infinite queueing system.

5) Number of service channels

It is a multi-channel system. The model consists of two infinite priority queue type A and B, and a set of c parallel, identical test systems. All the test systems are able to run all the test cases. The system is operated in such a way that at any time, any test case can be run by any test system. So upon arrival, a test case is assigned by one of the available test system, if any. If not, it must join one of the queues.

6) Number of service stages

The service is running the specified test case and getting the result. So it is a single-stage of service. As the loading procedure is very fast, we just ignore it.

So the behavior of the Scheduler can be viewed as a $M^X / M / c + M + G$(n) queueing system. The symbol M after the first + is to indicate the markovian assumption for reneging times. The G(n) after the second + is to indicate the balked rate which is a function of the number of the test cases in the waiting queue. The resulting model is shown on Figure 3-1, and will be referred to as basic model.

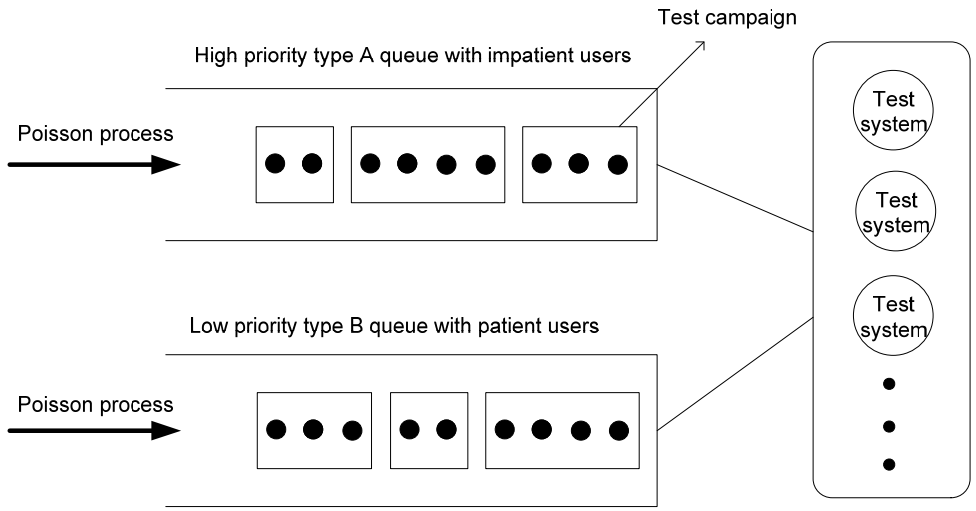


Figure 3-1: Basic Model

3.3 Simulation

The discrete-event stochastic simulation is used to analyze the Scheduler. There are three major elements in the simulation.

- Input distribution selection and generation
- Bookkeeping
- Output analysis

As we are interested in modeling stochastic systems, it is necessary to select and then generate the appropriate stochastic phenomena in the computer. We must decide on which probability distributions we wish to use to represent these arrivals and the service mechanisms. Then, random variates from these different distributions must be generated so that the system can be observed in action. Once these distributions are chosen and random variates are generated, the bookkeeping phase keeps track of the transactions moving around the system and keeps the counters on the ongoing processes in

order to calculate the appropriate performance measures. Output analysis has to do with statistical techniques required to make valid statements concerning system performance.

This methodology is used to simulate the Scheduler. In addition, Microsoft Visual C++ is adopted as the development environment. As we know, C++ is a programming language, optimal and with high speed floating point computation. But it is non-trivial to produce visual effects and difficult to secure a robust vector algebra package. While Matlab is an interpreted scripting language, has excellent prototyping and plotting functionality and contains convenient and very robust matrix operation packages. So "C++ invoking Matlab commands" technique is used to generate the specified distributions and produce visual effect result.

Figure 3-2 shows the simulation flow chart.

3.3.1 Input distribution

After collecting and analyzing the data of the following input elements, I am assuming they have the following distributions.

- Arrival rate of the campaigns, exponential distribution
- The size of a campaign, or the number of test cases in a campaign,

  uniform distribution
- Mean service time, or the duration of executing a test case,

  a minimum value + an exponential distribution variable
- Reneged time of the users,

  a minimum value + exponential distribution variable
- Balked rate of the users,

  it is a function of the number of the test cases in the waiting queue.
- Number of the test systems
- Queue discipline, either FCFS or PR (High priority or Low priority),

  assume 75% users have high priority and 25% users have low priority

3.3.2 Bookkeeping

As mentioned earlier, the bookkeeping phase of a simulation model must keep track of the transactions moving around the system, and set up the counters on the ongoing processes in order to calculate various measures of the system performance.

A master clock is advanced in a fixed increment of time. At every step, it first checks the reneged impatient users in the high priority queue. If it is the time that a user leaves, the high priority queue needs to be updated to remove all the test cases in the campaign that the user submitted. Next it checks the departure event. When the service timer is timed out on a test system, the correspondent test system status should be updated to idle. Last is checking the arrival event. If there is a coming user and the user is balked, continue the loop. Otherwise, add the test cases in the campaign that the user submitted to the queue based on the priority of the user. At the same time, check the availability of the test systems. If there are free test systems, transfer the test cases in the queues to the test systems. All the activities are logged in the log file.

3.3.3 Output analysis

In order to evaluate the effectiveness of the Scheduler, I record the following elements during the simulation. From the evaluation result, I know how many test systems is suitable for the demand and the relative fairness of the Scheduler.

- W_{qh}: Mean waiting time in the queue – high priority users
- L_{qh}: Mean number of the test cases in the queue – high priority users
- W_{ql}: Mean waiting time in the queue – low priority users
- L_{ql}: Mean number of the test cases in the queue – low priority users
- P_0: Fraction of time that the system is idle
- ρ: Service utilization, it equals to $1 - P_0$

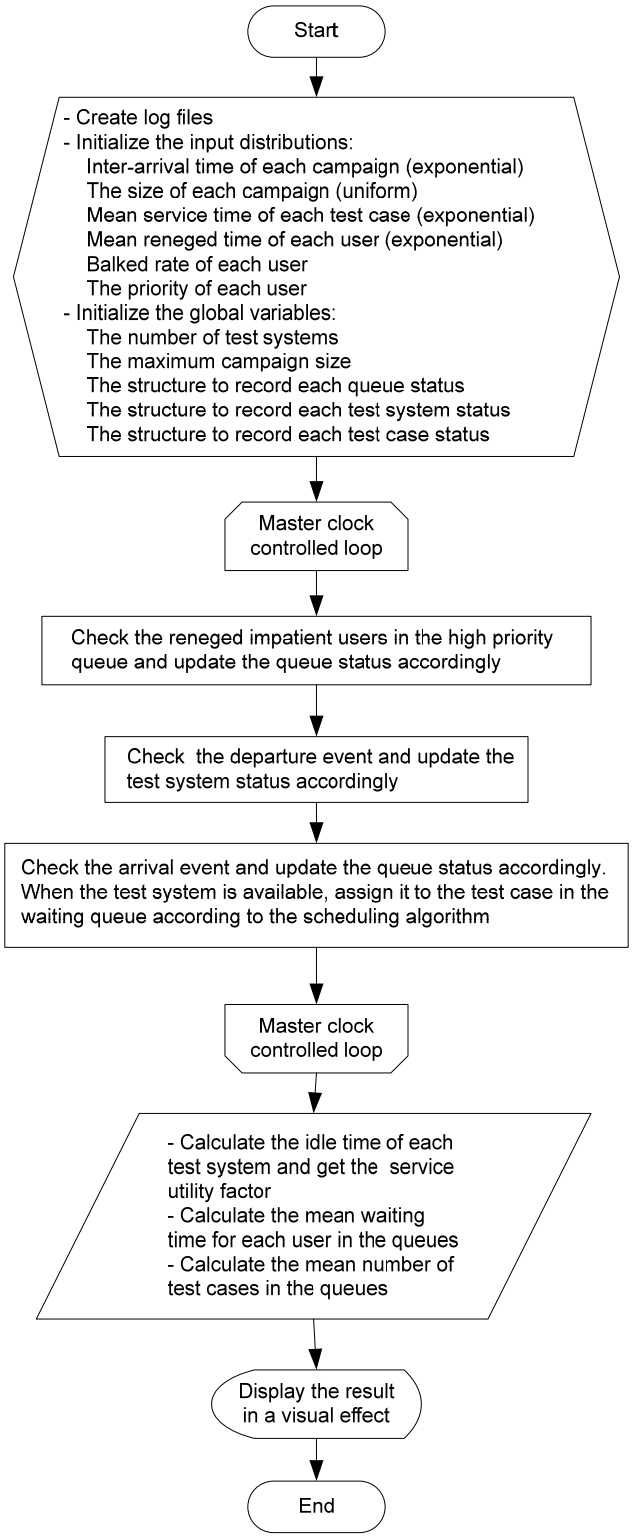


Figure 3-2: Simulation flow chart

3.4 Results

Table 3-1 lists the input data of the simulation. Table 3-2 lists the output data of the simulation with the scheduling algorithm I. Based on the basic model defined above, the test cases in queue A always have higher priority than those in queue B. Such that when the test system is available, the test cases in queue A can get the service first. In other word, only when the queue A is empty, the test cases in queue B can get the service. As the priority rule is non-preemptive, the coming high priority user can not stop the service of the low priority users. This is the scheduling algorithm I.

In Table 3-2, as the number of the test systems increases, the mean waiting time in the queues, the mean number of the test cases in the queues and the service utilization of the test systems all decrease. This is right from the common sense. According to the simulation result in Table 3-2, the framework needs at least eleven test systems.

But the test systems are very expensive resources. One set of the test system is around half million dollars. In order to save some test systems, I analyzed the data in Table 3-2 carefully. When the number of the test systems is below 11, the low priority users are starved – the waiting time is too long. To consider the fairness as well as the effectiveness, the new scheduling algorithm is demanded. As we know, “if the selection of service is no way a function of the relative size of the service time and if the average system size is unaltered, the average waiting time is the same. But the waiting time distribution will be changed because of the different queue discipline selection“. To increase the performance of the low priority users, the performance of the high priority users have to be sacrificed a little bit. The new scheduling algorithm adds a condition check in assigning the available test systems. When the queue B length is larger than a pre-defined value and the queue A length is less than a pre-defined value, assigning the available test system to the test cases in queue B instead of that in queue A. This is the scheduling algorithm II.

After comparing several simulation results with different pre-defined value pairs in the scheduling algorithm II, I found that the performance of the low priority users is all increased at various degrees and the performance of the high priority users is decreased accordingly. Table 3-2 lists the simulation output data with scheduling algorithm II (200, 100). (200, 100) means that when the queue B length is larger than 200 and queue A length is less than 100, the available test system is assigned to the test cases in queue B. The visual effect outputs are generated in figure 3-3-x. It is easy to identify the difference between the two scheduling algorithms. Using scheduling algorithm II, the mean waiting time and the mean number of TCs in the high priority queue A increase a little bit, but those parameters of the low priority users decrease a lot when the number of the test systems is less than 10. It is because we assume 75% percent of the incoming users have high priority and only 25% percent of them have low priority. When the number of the test systems is larger than or equal to 10, they are almost the same and the service utility of them becomes lower. It makes sense. Because the difference between these two algorithms is that the scheduling algorithm II adds a condition check when

assigning available test systems. Only when the length of queue B is large enough (>200) and at the same time the length of queue A is moderate (<100), the difference in scheduling algorithm II takes effect.

In conclusion, when the number of the test systems is limited, the scheduling algorithm II significantly increases the performance of the low priority users with the cost of decreasing the performance of the high priority users in a tolerable range. When there are enough test systems, the performance of both the high priority users and the low priority users is almost the same under these two scheduling algorithms. In addition, the scheduling algorithms almost do not affect the service utility of test system.

Table 3-1: Simulation input data table

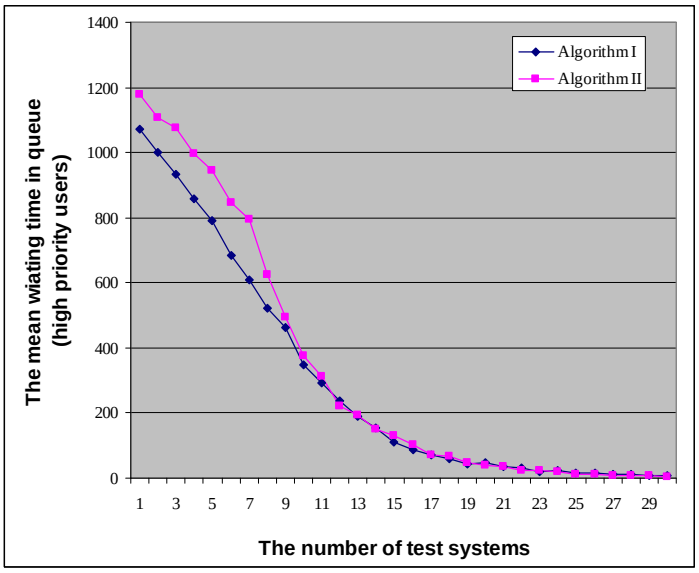
Input data	Value
Mean inter-arrival time	Exponential mean 20
Campaign size	Uniform over (1, 20)
Mean service time	5 + exponential mean 15
Reneged time (only for high priority users)	60 + exponential mean 30
Balked rate (only for high priority users)	When the queue length is larger than 100 , 20% users are balked
Number of test systems	1 - 30
Queue discipline	Two queues, one is for the high priority users and the other one is for the low priority users. 75% of the coming users are high priority and 25% of those are low priority. The priority rule is non-preemptive Inside each queue, it follows FCFS discipline.
The number of steps of the master clock	50000

Table 3-2: Simulation output data table – scheduling algorithm I

Test system number	W_{qh} (minutes)	W_{ql} (minutes)	L_{qh} (number of TCs)	L_{ql} (number of TCs)	ρ
1	1072.2	273926.81	39.08	3371.33	100%
2	1002.42	265404.54	36.96	3210.35	100%
3	933.46	250311.47	34.49	3097.78	100%
4	858.02	248262.48	31.63	3059.39	100%
5	791.6	223468.41	28.65	2720.88	100%
6	682.72	176119.02	25.27	2100.79	100%
7	609.54	154712.8	22.19	1775.55	100%
8	521.13	111261.19	18.88	1419.27	100%
9	464.61	39826.6	16.86	488.56	100%
10	349.67	5201.44	12.55	65.57	95%
11	294.33	2699.7	11.2	33.23	90%
12	239.03	1183.08	8.72	14.36	82%
13	190.63	1030.6	7.06	13.02	78%
14	152.42	534.92	5.64	6.74	72%
15	110.69	311.18	3.99	3.83	66%
16	86.82	287.28	3.16	3.44	63%
17	69.95	167.23	2.57	2.01	58%
18	61.27	136.19	2.29	1.59	56%
19	45.21	97.11	1.71	1.47	52%
20	45.89	96.46	1.67	1.21	50%
21	33.81	71.1	1.22	0.95	47%
22	30.53	65.35	1.17	0.8	47%
23	20.72	52.2	1.05	0.68	43%
24	24.17	45.06	0.92	0.55	43%
25	17.1	24.79	0.63	0.3	41%
26	13.85	20.01	0.5	0.25	38%
27	11.64	14.22	0.42	0.18	37%
28	10.8	13.71	0.4	0.17	36%
29	8.54	12.71	0.32	0.15	35%
30	6.6	8.76	0.24	0.11	33%

Table 3-3: Simulation output data table – scheduling algorithm II

Test system number	W_{qh} (minutes)	W_{ql} (minutes)	L_{qh} (number of TCs)	L_{ql} (number of TCs)	ρ
1	1177.42	161824.39	44.4	1886.23	100%
2	1107.24	66504.85	41.2	814.1	100%
3	1076.22	18302.43	40	234.83	100%
4	995.82	16519.07	37.3	209.41	100%
5	945.28	16973.85	34.4	199.95	100%
6	847.17	15507.76	31.3	196.73	100%
7	796.3	15038.59	28.5	193.4	100%
8	623.34	14375.68	23	176.11	100%
9	492.55	10112.71	18.1	122.47	99%
10	376.39	5580.01	13.8	69.1	96%
11	311.5	3117.84	11.7	37.47	91%
12	223.13	1327.84	8.21	16.58	83%
13	194.16	748.91	7.18	9.28	78%
14	151.29	600.13	5.58	7.78	73%
15	130.24	446.5	4.84	5.42	68%
16	101.77	245.89	3.75	3.01	63%
17	72.35	215.87	2.62	2.62	58%
18	68.18	170.25	2.61	2.13	59%
19	47.42	106.41	1.76	1.32	53%
20	39.21	80.89	1.41	0.95	49%
21	35.72	75.57	1.33	0.91	48%
22	25.28	55.71	0.93	0.69	46%
23	22.49	37.82	0.82	0.45	43%
24	18.09	35.04	0.69	0.42	42%
25	13.78	22.68	0.51	0.29	39%
26	12.78	18.22	0.46	0.21	38%
27	9.68	17.32	0.38	0.21	37%
28	9.77	16.5	0.36	0.17	37%
29	8.71	15.55	0.33	0.15	36%
30	5.85	6.67	0.21	0.08	33%



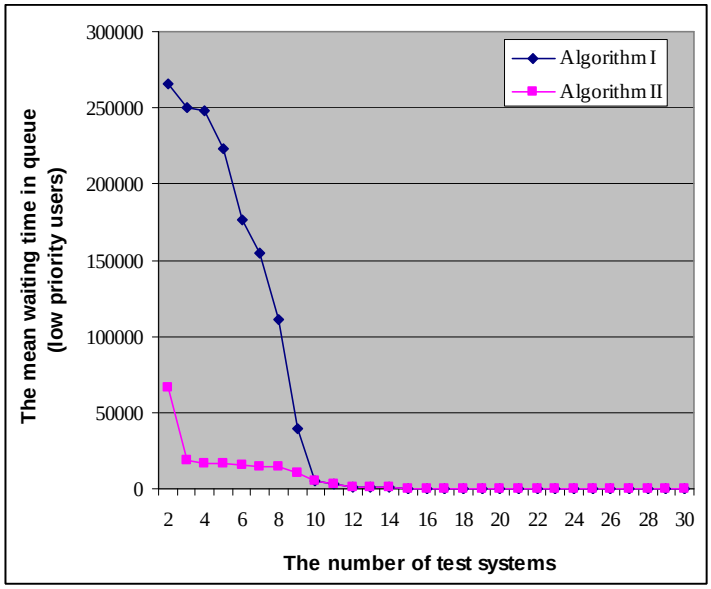
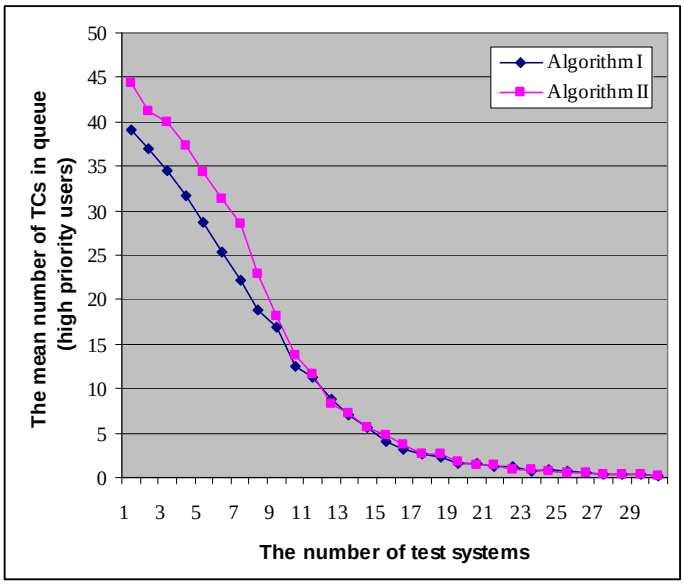


Figure 3-3-1: The mean waiting time in queue vs. the number of test systems



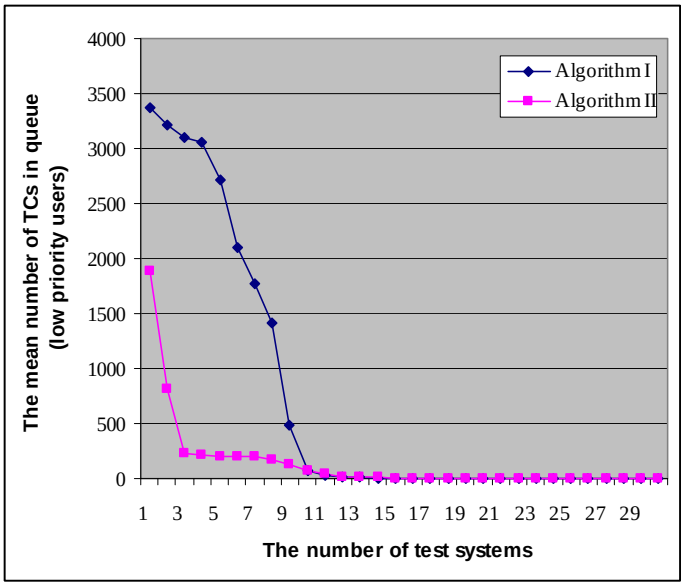


Figure 3-3-2: The mean number of TCs in queue vs. the number of test systems

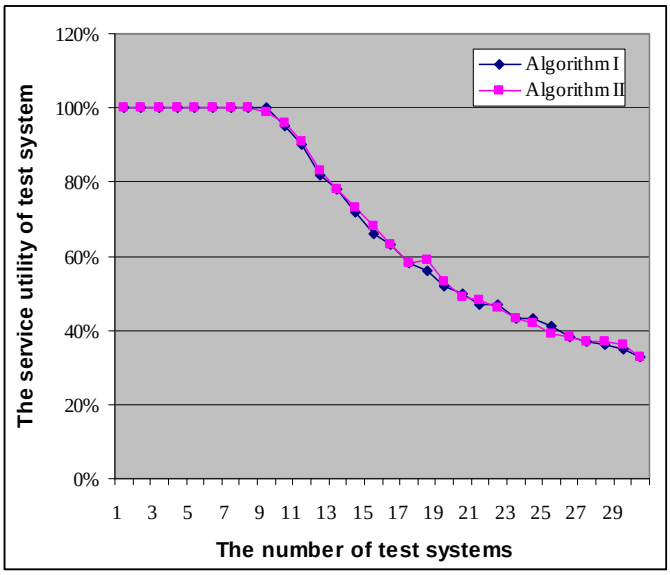


Figure 3-3-3: The service utility vs. the number of test systems

Chapter 4
Test Engine Design

Test Engine is the executive of the test framework. It consists of two major software components. They are Test Manager and Automator. Besides these, there are two assistant hardware components, Switch for RF/Power/USB and Fake Battery. This chapter will focus on the design of Test Manager and Automator.

4.1 Test Manager

The Test Manager resides in the Controller PC of each test system. The main task of the Test Manager is controlling the communication between the Scheduler and the Automator, taking control of launching test cases on the test equipment, error monitor and recovery.

4.1.1 The console design of the Test Manager

Figure 4-1 is the console of the Test Manager. There are two operation modes, local control and remote control. In local control mode, the Test Manager does not connect to the Scheduler and only talks to the Automator. The tester configures the test equipment via console. Developers can do the component test and debugging in this mode. The remote control mode is for the automated test framework. The Test Manager configures the test equipment automatically as per the information that the Scheduler requests. The console consists of the following four parts.

The top area displays the current test equipment name, operation mode and the Test Manager version. The following four control buttons are for connecting to the Automator, disconnecting from the Automator, resetting the test equipment and SUT. The other three triggering buttons are for testing purpose.

The “Platform Independent” part is for all the test equipments. It selects or shows the current running project name, test equipment name, the IMEISV (International Mobile Equipment Identity and Software Version) and IMSI (International Mobile Subscriber Identity) values of the SUT. The other three check boxes are for controlling purpose.

The “Platform Dependent” part is mainly used in the local control mode. The content of this part is controlled by the value of the “Test Sub Ctrl” in the “Platform Independent” part. The user can

generate and load the test campaign, select the running type, set the result path, launch or stop test campaign and view logs for the selected test equipment.

The last part is the event log window. It records all the activities that the Test Manager is undergoing. It uses different colors for different kind of the logs. The user can save or clear the logs in the local control mode. In the remote control mode, the event logs are saved to the specified files per hour basis. These logs are very useful for the debugging.

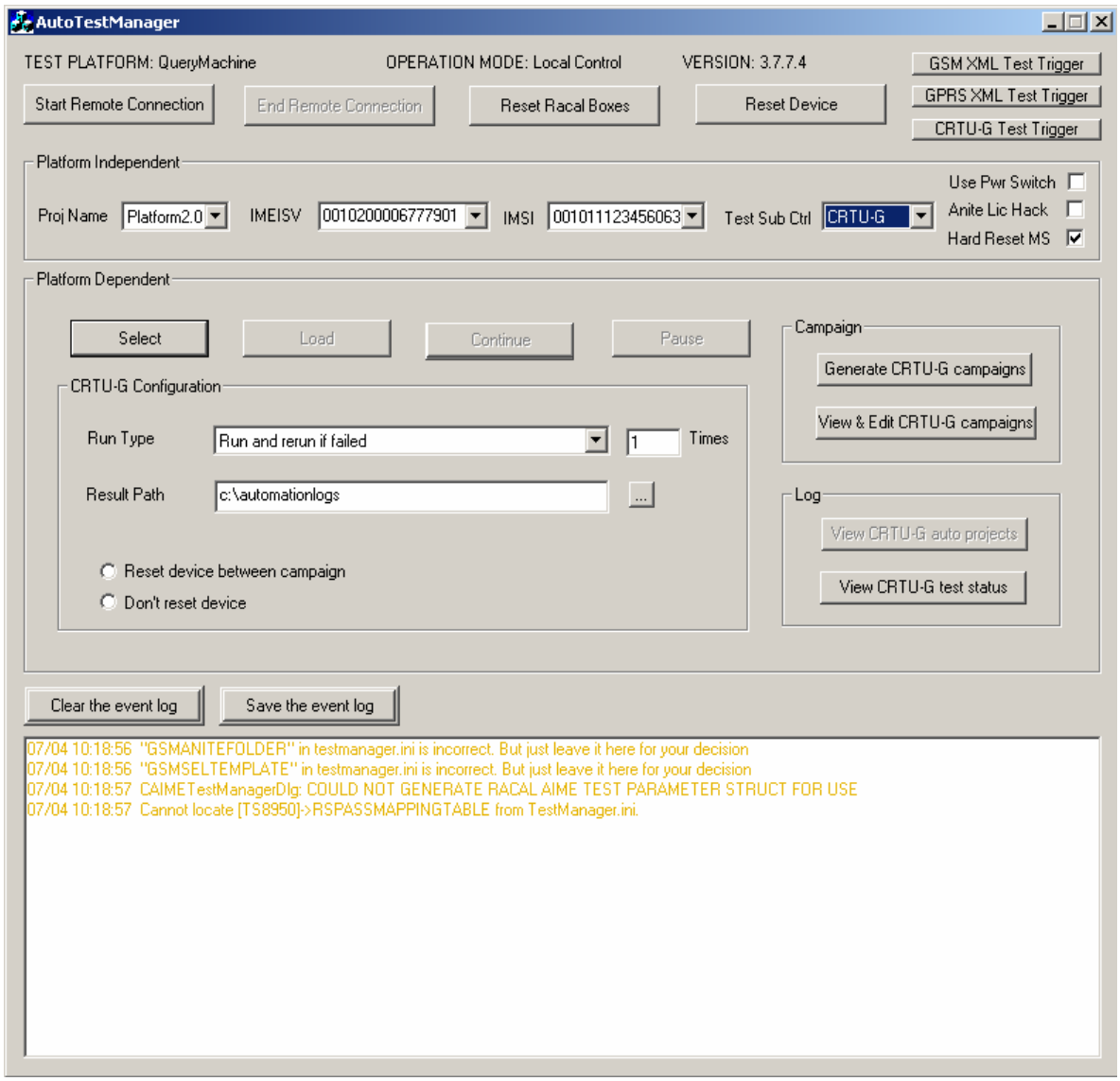


Figure 4-1: Test Manager Console

4.1.2 The state machine design of the Test Manager

In order to handle the messages between the Scheduler and the Automator correctly and effectively, a state machine is developed in the Test Manager. Figure 4-2 displays this state machine. There are 5 states defined in the state machine. They are idle state, ready state, loading state, running state and resetting state.

- Idle state:

The Test Engine is inactive. The Scheduler, Test Manager and Automator are in the stand-alone mode. There are no connections among them.

When the Test Manager is in Ready, Running or Resetting states and receives the ACTION_TERMINATE command from the Automator, the Test Manager enters idle state. Besides this, when the Test Manager is in Loading state and receives the INICATION_LOAD from the Automator with the loading result for LOADDEFAULT is failed, the Test manger also enters Idle state.

- Ready state:

The Test Engine is active. The Scheduler, Test Manager and Automator are connected. But there is no specific undergoing task.

- Loading state:

This state is triggered by the ACTION_LOAD / ACTION_LOADDEFAULT commands sent by the Scheduler. After the Test Manager receives this command, it forwards the command to the Automator, starts the loading guard timer and enters the Loading state.

- Running state:

This state is triggered by the INDICATION_EXECUTE command sent by the Automator. After the Test Manager receives the ACTION_EXECUTE command from the Scheduler, the Test Manager forwards this command to the Automator. Then the Automator gets the device IMEISV value, composes the INDICATION_EXECUTE command and sends it to the Test Manager. At this time, the Test Manager launches the test case and enters Running state.

- Resetting state:

This state is triggered by the INDICATION_USBREMOVE command sent by the Automator. In this state, the SUT is resetting for some reasons. After the SUT is recovered from the resetting, the Automator sends INDICATION_USBREADY command to inform the Test Manger that the SUT is

ready for testing. Then the Test Manager enters Ready state. Otherwise, the Test Manager gets ACTION_TERMINATE command which means the SUT encounters an unrecoverable issue and the Test Manager enters idle state.

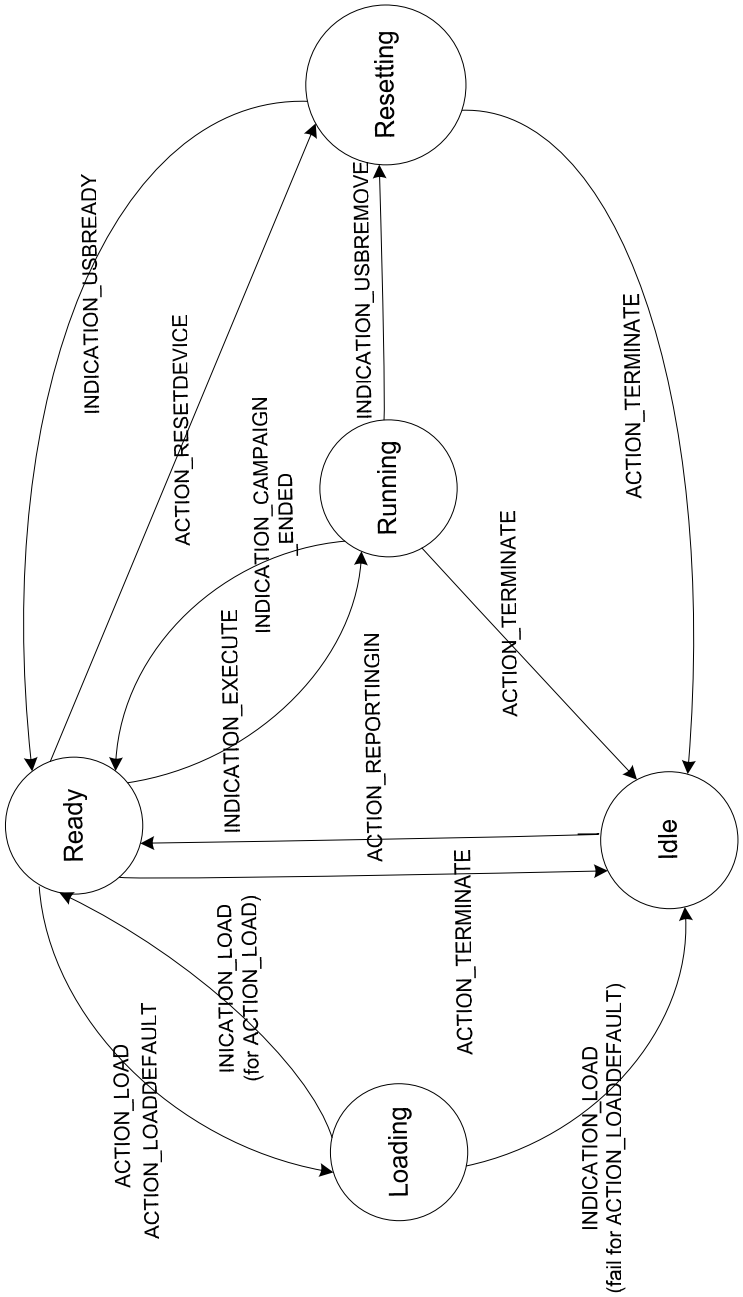


Figure 4-2: The state machine of the Test Manager

4.2 Automator

The Automator resides in the Controller PC of each test system. The main task of the Automator is manipulating the SUT according to the commands sent from the test equipment, taking test log and result, controlling the switch.

4.2.1 The console design of the Automator

There are three panels in the Automator console. They are the Control Panel, Log Panel and Result Panel. Control Panel includes the Device Control, Port Control and others.

- Device Control

The Device Control window includes the project name, the SUT build version and bundle, testing band and type, the SUT type and the IMEI, IMEISV values. This information can be updated automatically in the automatic test mode.

- Port Control

The Port Control window configures the ports that connect to the SUT and the test equipment. For example, the SUT is connected to the Auotmator via USB #1 and the test equipment is via RS-232 serial port with the baud rate 115200 bps. The ping-pong button “Disconnect / Connect” is for disconnecting from / connecting to the SUT. The “Reset Device” button is for resetting the SUT.

- Others

- The Operator is used to record the tester’s identity.
- The Platform is for displaying the current test equipment name.
- The Simulator check box is for testing purpose. When it is checked, the Automator is in the test mode. Usually, it is unchecked.
- When the NoLoad checkbox is checked, the loading procedure is skipped. It is assumed that the SUT is pre-loaded. Usually, it is unchecked.
- When the Switch is in use, the Switch checkbox is checked and the “Switch Control” button is enabled. All the connected SUT information can be obtained via “Switch Control” button.
- The Power Switch is a programmable instrument. In the automation test framework, all the power suppliers of the test equipments are under the control of the Power Switch. It connects to the Automator via TCP/IP. The IP address and port are configured in the Control Panel.

Log Panel records all the automator’s activities. The AT commands sent to the SUT by the Automator are displayed at the left bottom. The developers can type the AT command in the edit box

and click Execute button to test the AT command. Result Panel shows the test case result and the duration. All the logs and results are saved in the specified “Test Result” file.

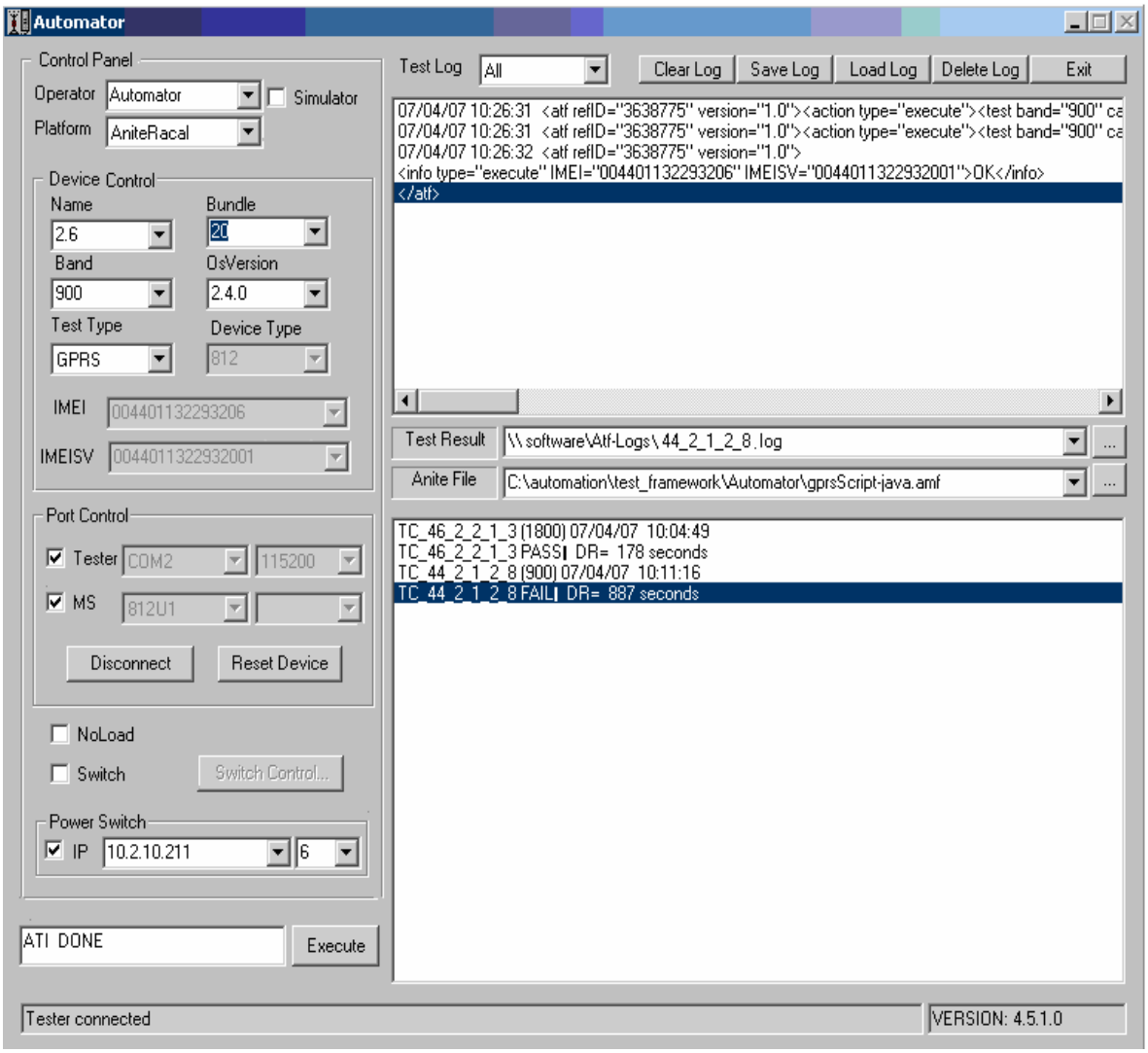


Figure 4-3: Automator Console

4.2.2 The Automator architecture

There are three main components that Automator interacts. Consequently, three main threads are designed to either control or manage the three components. They are working parallel. The communication among the threads is the window messages. Figure 4-4 shows the architecture.

4.2.2.1 State Machine

This is the Automator main thread. It is the interface to the Test Manager. It performs the light weight tasks, such as receiving and transferring messages from or to the other threads. It is supposed to be never blocked. So that the Automator is always ready to respond to the messages.

4.2.2.2 SUT Controller

This thread controls all the SUTs connected to the test system. It is the interface to the SUTs. It talks with the power / RF / USB switch to control the SUTs. It communicates with the State Machine to identify and select the SUTs. It also talks with the Message Handler to get the commands from the test equipment and manipulate the SUTs during test.

4.2.2.3 Message Handler

This is the thread that listens to the RS-232 serial port of the test system, converts the received text messages to the commands that SUT understands, sends the commands to the SUT Controller thread, waits the response from the SUT Controller and sends the response to the test system. It is the interface to the test system. The message buffer needs to be implemented and well managed.

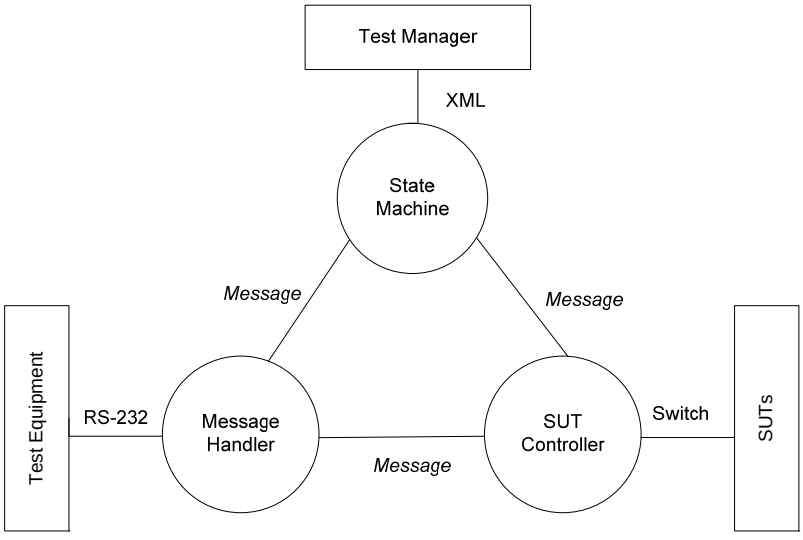


Figure 4-4: Automator architecture

4.2.3 Message Handler design

Message Handler is converting the message between the test system and the SUT. It is an important component in the Automator. It needs five different functions.

- Listen for messages from the test system controller PC
- Translate the message from the test system controller PC into a message that the SUT understands
- Send a response to the test system controller PC
- Send a message to the SUT
- Listen for messages from the SUT

4.2.3.1 The messages from the test system

One of the Message Handler's tasks is listening the messages from the test system controller PC. The messages from the controller PC are in fixed format, such as "Close this message box, then answer the call at the Mobile".

There are two operation modes in the test system, manual mode and automation mode. When it is in the automation mode, the messages are sent to the RS-232 serial port of the controller PC instead of popping up a message box.

4.2.3.2 The messages that the SUT understands

Once the Message Handler catches a message, it translates the message into the format that the SUT understands. AT command is widely used in the telecommunication area. Most SUT manufacturers implement the AT command parser in the SUT.

AT command is a string of characters sent to the SUT while the SUT is in a command state. A command line has a prefix, a body and terminator. Each command line must begin with the character sequence AT and must be terminated by a carriage return. Command entered in upper case or lower case is accepted, but both the A and T must be of the same case, i.e. "AT" or "at". The default terminator is the ENTER key <CR> character. Characters that precede the AT prefix are ignored. The command line interpretation begins upon receipt of the ENTER key character. Characters within the command line are parsed as commands with associated parameter values. The basic commands consist of single ASCII characters, or single characters proceeded by a prefix character following by a decimal parameter. Here are the examples.

Message from test equipment	AT Command
Initiate a Mobile Originating call for basic service C_Telephony	ATD123456<CR>
Answer the call at the Mobile	ATA<CR>
Set the MS to perform GPRS Attach	AT+CGATT=1<CR>

4.2.3.3 The program design

The messages from the test system controller PC are sent in a fixed format and can be mapped to AT commands that the SUT understands. We also consider the future changes, such as adding new messages, removing old unused messages and updating the existing messages. So a message file is designed. These are the rules for the file format.

- There are three lines for each message. They are message from the test system, response to the test system and the message to the SUT.
- Each set of messages is preceded by a line containing six asterisks
- Lines which begin with two forward slash are comments

Here is the example of a message file.

//

// message file example

//

******

Close this message box, then answer the call at the Mobile
RESPONSE:ID_OK
ATA
******
Close this message box, then initiate a Mobile Originating call for basic service C_Telephony
RESPONSE:ID_OK
ATD123456
******
Set the MS to perform GPRS Attach
RESPONSE :ID_YES
AT+CGATT=1

Figure 4-5 shows the flow chart of the Message Handler. A thread is monitoring the output of RS-232 serial port on the test system controller PC. Once it gets a message. It scans the predefined message file to match the message string. If there is no match, the message handler logs and displays the error and sends the default response OK to the controller PC. It is a defect of the software. The developer will add the message into the message file after geting the error. If it finds the match, it is easy to get the corresponding response ID and AT command. The response ID is sent to the controller PC to let it know that the message has been received. The AT command is sent to the SUT controller thread. Then the SUT controller thread adds it to the tail of the queue which is used to store the incoming AT commands. The AT command at the header of the queue is sent to the SUT when it is ready. The next AT command at the queue header is sent to the SUT when it gets the OK response from the SUT for the previous command. If the timeout expires when waiting for the response or the response is not OK sequentially three times, an error handler will be invoked.

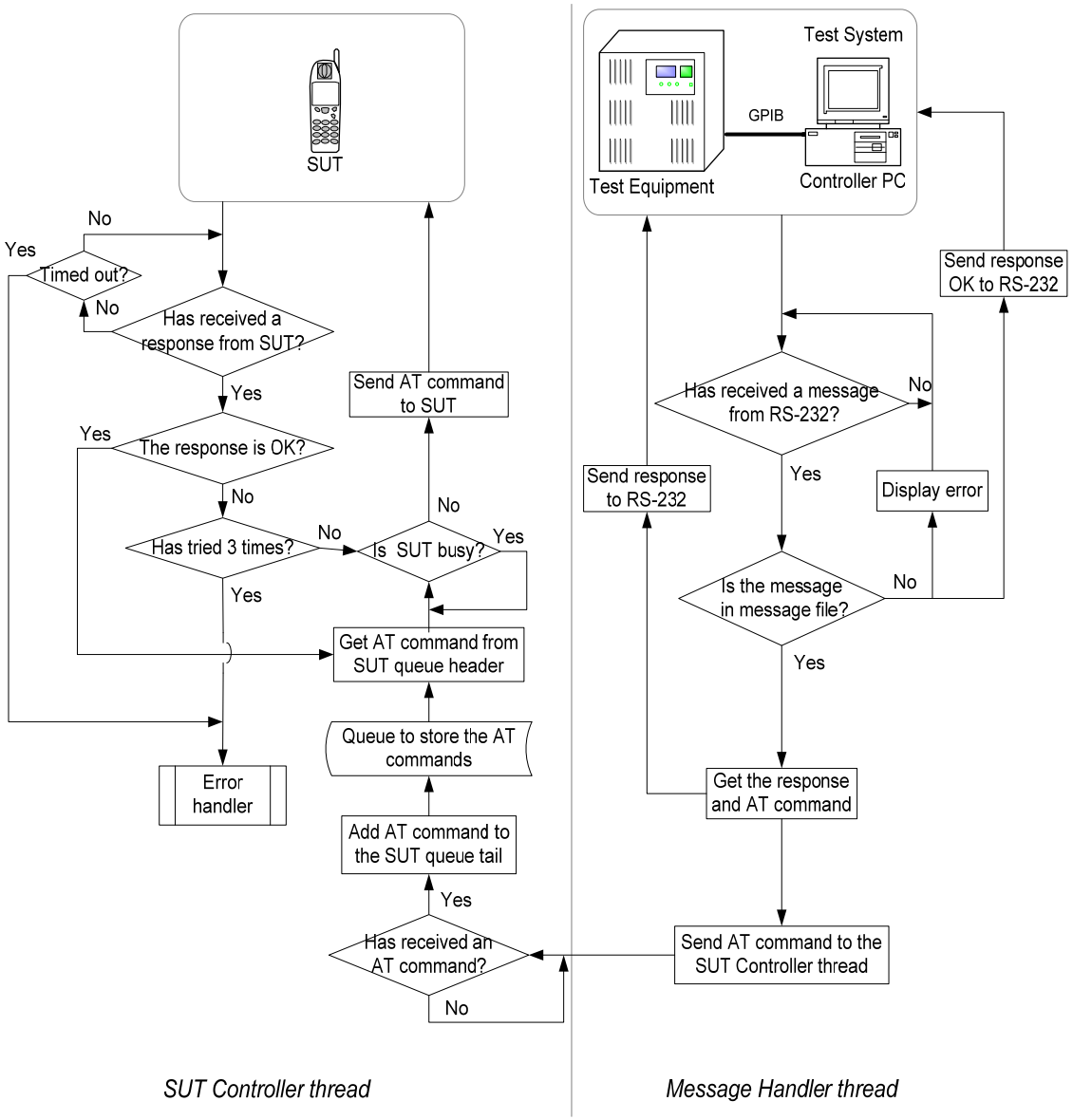


Figure 4-5: Message Handler flow chart

Chapter 5
Conclusion

The primary objective of this thesis is to introduce the design of the automated test framework for the wireless protocol stack development. There are four major components in the automated test framework. They are the Front End, Scheduler, Test Engine and Data Storage. The architecture of the framework and the function of each component are introduced. Then, the protocol design among the Front End, Scheduler and Test Engine are described.

Scheduler is the central controller of the framework. The scheduling algorithm affects the performance of the system. In this thesis, a $M^X / M / c + M + G$ (n) queueing system is modeled and simulated according to the queueing theory. Based on the simulation result, a better scheduling algorithm is proposed. It considers the fairness as well as the effectiveness. With this new scheduling algorithm, the performance of the low priority users gets improved when the number of the test systems is limited.

Test Engine is the executive of the framework. Test Manager and Automator are the two major components of the Test Engine. In this thesis, the detail design of these two components is introduced. It includes the component architecture design, console design, state machine design and the flow charts of some components.

Finally, I would like to describe the current situation of the automated test framework. It covers around 90% of the test cases. Both the developers and testers are satisfied with the performance of the system. It has the capability to work around the clock. The deployment of the automated test framework tremendously boosts the wireless protocol stack development. In the future, the test framework will be expanded to include different kind of the test engines, like the SIM test systems, RF test systems and so on. As the new technologies are emerging, the test framework will be updated to support them definitely.

Bibliography

1. Sheldon M. Ross, 2003, “Introduction to Probability Models, 8th edition”, Published by Academic Press
2. Donald G., 1998, “Fundamentals of Queuing Theory”, Published by John Wiley & Sons Inc.
3. Kanglin Li, 2004, “Effective Software Test Automation: Developing an Automated Software Testing Tool”, Published by Addison-Wesley Professional
4. Schwartz M., 1996, “Broadband integrated networks”, Published by Prentice Hall PTR
5. Jiantao Pan, 1999, “Software Testing”, Carnegie Mellon University
6. Erik Ray, 2003, “Learning XML”, Published by O’Reilly
7. Bruce Eckel, 2000, “Thinking in C++”
8. Jeffrey Richter, 1999, “Programming Applications for Microsoft Windows”, Published by Microsoft
9. Per Brinch Hansen, 2001, “Operating System Principle”, Published by Prentice Hall
10. ETSI TC-MTS, 1995, “Methods for Testing and Specification (MTS); Protocol and profile conformance testing specifications; Standardization methodology”
11. ETSI TS 151 010-1 v7.5.0, 2007, “Digital cellular telecommunications system (phase 2+), Mobile Station conformance specification, Part 1: Conformance specification”
12. ETSI GSM 07.07, “AT Command set for GSM Mobile Equipment”
13. ETSI, ”Making Better Standards”, Published on http://portal.etsi.org/mbs/Testing/Comparison/comparison.asp
14. Sungwon Kang, 1998, “Relating interoperability testing with conformance testing”
15. Dibuz, S. & Kremer P., 2006, “An easy way to test interoperability and conformance”
16. Venkatesh K. B., 1990, “Conformance Testing in the Telecommunications Industry”

17. Muhammad, K., 2004, “Scheduling Algorithms for HS-DSCH in a WCDMA Mixed Traffic Scenario”, Published on The 14th IEEE 2003 International Symposium on Personal, Indoor and Mobile Radio Communication Proceedings
18. Anite, “Writing an automation program for CT (GSM) or the CT (EGPRS) Test Manager”
19. Anite, “RCMI User Guide”
20. “ATF Project Requirement” & “ATF Design Specification”
21. Oualid Jouini & Yves Dallery, 2006, “Predicting Queueing Delays for Multiclass Call Centers”
22. Daniel Nurmi, 2006, “Evaluation of a Workflow Scheduler Using Integrated Performance Modelling and Batch Queue Wait Time Prediction”
23. Erol A, Pekoz, 2002, “Optimal Policies for Multi-server Non-preemptive Priority Queues”
24. J.Kay & P.Lauder, 1988, “A Fair Share Scheduler”
25. Andreas Brandt & Manfred Brandt, 1998, “On a Two-Queue Priority System with Impatient and its Application to a Call Center”
26. H. Christian Gromoll, 2006, “The Impact of Reneging in Processor Sharing Queues”